Guida alla programmazione

Borland®
C++Builder™ 6
per Windows

Borland Software Corporation, 100 Enterprise Way
Scotts Valley, CA 95066-3249

Per una lista completa dei file che è possibile distribuire in conformità all'Accordo di Licenza e Garanzia Limitata di C++Builder, fare riferimento al file DEPLOY.TXT situato nella directory principale di C++Builder 6.

Borland potrebbe avere brevetti e/o applicazioni in corso di brevetto relativamente ad argomenti trattati in questo documento. La fornitura di questo documento non conferisce alcuna licenza relativamente a questi brevetti. Per un elenco dei brevetti in vigore, fare riferimento al CD del prodotto oppure alla finestra di dialogo About.

Stampato in Irlanda

CPE1340WW21001 4E0R1001
0102030405-9 8 7 6 5 4 3 2 1
D3

Indice generale

Capitolo 5
Operazioni con componenti 5-1

Capitolo 6
Uso dei controlli 6-1

Capitolo 7
Creazione di applicazioni, componenti e librerie 7-1

Capitolo 8
Sviluppo dell'interfaccia utente dell'applicazione 8-1

Capitolo 9
Tipi di controlli 9-1

Capitolo 13
Supporto del linguaggio C++ per la VCL e la CLX 13-1

Capitolo 14
Sviluppo di applicazioni multipiattaforma 14-1

Capitolo 15
Uso di package e di componenti 15-1

Capitolo 16
Creazione di applicazioni internazionali 16-1

Capitolo 17
Distribuzione di applicazioni 17-1

Parte II
Sviluppo di applicazioni database

Capitolo 18
Progettazione di applicazioni database 18-1

Capitolo 19
Uso dei controlli dati 19-1

Capitolo 20
Uso dei componenti di supporto decisionale 20-1

Capitolo 21
Connessione ai database 21-1

Capitolo 22
I dataset 22-1

Capitolo 23
Operazioni con i componenti campo 23-1

Capitolo 24
Uso di Borland Database Engine 24-1

Capitolo 25
Operazioni con componenti ADO 25-1

Capitolo 26
Uso di dataset unidirezionali 26-1

Capitolo 27
Utilizzo dei dataset client 27-1

Capitolo 28
Uso di componenti provider 28-1

Capitolo 29
Creazione di applicazioni multi-tier 29-1

Capitolo 30
Uso di XML nelle applicazioni database 30-1

Parte III
Creazione di applicazioni Internet

Capitolo 31
Scrittura di applicazioni CORBA 31-1

Capitolo 32
Creazione di applicazioni server per Internet 32-1

Capitolo 33
Utilizzo di Web Broker 33-1

Capitolo 34
Creazione di applicazioni Web Server con WebSnap 34-1

Capitolo 35
Operazioni con documenti XML 35-1

Capitolo 36
Utilizzo dei Web Services 36-1

Capitolo 37
Operazioni con i socket 37-1

Parte IV
Sviluppo di applicazioni COM

Capitolo 38
Panoramica sulle tecnologie COM 38-1

Capitolo 39
Funzionamento delle librerie di tipi 39-1

Capitolo 40
Creazione di client COM 40-1

Capitolo 41
Creazione di semplici server COM 41-1

Capitolo 42
Creazione di Active Server Page 42-1

Capitolo 43
Creazione di un controllo ActiveX 43-1

Capitolo 44
Creazione di oggetti MTS o COM+ 44-1

Parte V
Creazione di componenti custom

Capitolo 45
Introduzione alla creazione di componenti 45-1

Capitolo 51
Gestione dei messaggi e delle notifiche di sistema 51-1

Capitolo 52
Come rendere disponibili i componenti in fase di progetto 52-1

Tabelle

Figure

Capitolo

1

Introduzione

Il manuale *Guida alla programmazione* descrive gli argomenti, intermedi e evoluti, quali la costruzione di database client/server, la scrittura di componenti custom, la creazione di applicazioni per Web server Internet e l'inclusione del supporto per specifiche standard come SOAP, TCP/IP, COM e ActiveX. La maggior parte delle funzioni evolute che supportano lo sviluppo per Web, le tecnologie evolute XML e lo sviluppo di database richiedono componenti o wizard che non sono disponibili in tutte le versioni di C++Builder

Il manuale *Guida alla programmazione* presuppone una certa familiarità con l'uso di C++Builder e la conoscenza delle tecniche fondamentali di programmazione in C++Builder. Per una introduzione alla programmazione in C++Builder e all'ambiente di sviluppo integrato (IDE), consultare il manuale *Per iniziare* e la Guida in linea.

Contenuto del manuale

Questo manuale è costituito da cinque parti:

- La **Parte I, "Programmazione con C++Builder",** descrive come costruire applicazioni C++Builder per scopi generici. Vi vengono fornite informazioni dettagliate sulle tecniche di programmazione, utilizzabili in qualsiasi applicazione C++Builder. Per esempio, vi viene descritto l+uso di oggetti della Visual Component Library (VCL) comuni o della Component Library for Cross-platform (CLX), che semplificano la programmazione dell'interfaccia utente, per aspetti quali la gestione delle stringhe, la manipolazione del testo, l'implementazione delle finestre di dialogo comuni di Windows, le operazioni con elementi grafici, la gestione degli errori e delle eccezioni, l'uso di DLL, di Ole Automation e la scrittura di applicazioni internazionali.

  Capita di rado che una libreria VCL di C++Builder sia scritta in Object Pascal. Tuttavia, ci sono casi in cui ciò vale anche per programmi C++Builder. Un capitolo sul supporto del linguaggio C++ e sulle librerie VCL descrive alcuni argomenti

relativi al linguaggio quali le diverse modalità con cui si istanzia una classe C++ quando si utilizzano le classi VCL o le estensioni del linguaggio C++ aggiunte per supportare il modello di programmazione “componente-proprietà-evento” di C++Builder.

Un capitolo descrive come utilizzare gli oggetti inclusi nella Borland Component Library for Cross-Platform (CLX) per lo sviluppo di applicazioni che possono essere compilate ed eseguite su piattaforme Windows o Linux.

Il capitolo sulla distribuzione illustra dettagliatamente le operazioni connesse alla distribuzione delle applicazioni agli utenti. Per esempio, include informazioni sulle opzioni del compilatore, sull’uso di InstallShield Express, sulla concessione delle licenze e su come determinare quali package, DLL e altre librerie utilizzare per generare una versione della propria applicazione da immettere sul mercato.

- La **Parte II, “Sviluppo di applicazioni database”,** descrive come costruire applicazioni database utilizzando strumenti e componenti database. C++Builder permette di accedere a molti tipi di database, inclusi i database locali come Paradox e dBASE e database SQL server per rete come InterBase, Oracle e Sybase. È possibile scegliere fra vari meccanismi per l’accesso di dati, fra cui dbExpress, Borland Database Engine, InterBaseExpress e ActiveX Data Objects (ADO). Per implementare applicazioni database client/server più evolute, sono necessarie funzionalità di **C++Builder** che non sono disponibili in tutte le edizioni.
- La **Parte III, “Creazione di applicazioni Internet”,** descrive come creare applicazioni da distribuire in Internet. C++Builder include una vasta gamma di strumenti per la scrittura di applicazioni per Web server, fra cui Web Broker, un’architettura che permette di creare applicazioni server per più piattaforme, WebSnap, che permette di progettare pagine web in un ambiente GUI, il supporto per operazioni con documenti XML, e BizSnap, un’architettura per l’utilizzo di Servizi Web basati su SOAP.

  Questa parte contiene anche un capitolo sui componenti socket di C++Builder che permettono di creare applicazioni in grado di comunicare con altri sistemi tramite TCP/IP e i protocolli correlati. I socket forniscono connessioni basate sul protocollo TCP/IP, ma sufficientemente generiche da consentire di lavorare con protocolli correlati come Xerox Network System (XNS), DECnet di Digital o la famiglia IPX/SPX di Novell.
- La **Parte IV, “Sviluppo di applicazioni COM”,** descrive come costruire applicazioni in grado di interagire con altri oggetti API basati su COM. C++Builder supporta le applicazioni COM basate sui wizard ATL (Active Template Library) e un Type Library editor per semplificare lo sviluppo di server COM, oltre a uno strumento di importazione che consente di creare rapidamente applicazioni client. Il supporto per i client COM è previsto in tutte le versioni di C++Builder. Il supporto per i server COM non è disponibile in tutte le versioni di C++Builder.
- La **Parte V, “Creazione di componenti custom”,** descrive come progettare e implementare componenti personali e come renderli disponibili nella Component palette dell’IDE. Un componente può essere un qualsiasi elemento di programma che si desidera gestire in fase di progettazione. L’implementazione di componenti

personalizzati implica la derivazione di una nuova classe a partire da uno dei tipi di classe presenti nelle librerie di classi VCL o CLX.

Convenzioni tipografiche

Il presente manuale utilizza i caratteri e i simboli descritti nella Tabella 1.1 per indicare un testo speciale.

Tabella 1.1 Caratteri tipografici e simboli

Carattere o simbolo	Significato
`Monospaziato`	Il carattere monospaziato viene utilizzato per rappresentare un testo così come appare sullo schermo o nel codice C++. Rappresenta, inoltre, tutto ciò che deve essere immesso da tastiera.
[ ]	Le parentesi quadre nel testo o nei listati sintattici racchiudono elementi facolativi. Un testo di questo tipo non deve essere scritto letteralmente.
Grassetto	Le parole in grassetto nel testo o nei listati di codice rappresentano parole riservate di C++ oppure opzioni del compilatore.
Corsivo	Le parole in corsivo nel testo rappresentano identificatori di C++, come i nomi di variabili o di tipi. Il corsivo viene anche utilizzato per mettere in evidenza determinate parole, come i nuovi termini.
Keycaps	Questo carattere indica un tasto sulla tastiera. Ad esempio, "Premere *Esc* per uscire dal menu".

Servizi di assistenza agli sviluppatori

Borland offre diverse opzioni di supporto. Tra queste vi sono servizi gratuiti su Internet, dove si possono fare ricerche in un'ampia base di informazioni o collegarsi con altri utenti dei prodotti Borland. Inoltre, è possibile scegliere tra varie categorie di supporto, che vanno dal supporto all'installazione del prodotto Borland al supporto a livello di consulenza e all'assistenza estesa non gratuiti.

Per ottenere informazioni sui costi dei servizi Borland di assistenza agli sviluppatori, visitare il sito Web di Borland, all'indirizzo http://www.borland.com/devsupport/bcppbuilder, oppure all'indirizzo htpp://www.borland.it (seguire i collegamenti a Borland Assist), oppure contattare telefonicamente Borland Assist al numero 0542.34058. I clienti non residenti negli Stati Uniti d'America possono anche consultare il sito web di Borland all'indirizzo http://www.borland.com/bww/intlcust.html.

Quando si contatta il supporto tecnico, è bene avere sotto mano tutte le informazioni relative all'ambiente, alla versione del prodotto utilizzato e una descrizione dettagliata del problema.

Come ordinare la documentazione cartacea

Per ordinare la documentazione aggiuntiva, visitare il sito http://shop.borland.com.

Parte

I

Programmazione con C++Builder

I capitoli inclusi nella sezione "Programmazione con C++Builder" illustrano i concetti di base e le nozioni necessarie per creare applicazioni C++Builder con qualunque edizione del prodotto.

Capitolo

2

Sviluppo di applicazioni con C++Builder

Borland C++Builder è un ambiente di programmazione visuale orientato agli oggetti per lo sviluppo di applicazioni a 32-bit da distribuire in ambiente Windows e Linux. Con C++Builder è possibile creare applicazioni ad elevata efficienza scrivendo pochissime righe di codice.

C++Builder mette a disposizione una suite di strumenti di progettazione RAD (Rapid Application Development) che include wizard per la programmazione oltre a modelli di applicazioni e schede, e supporta la programmazione orientata agli oggetti grazie a una completa libreria di classi:

- la *Visual Component Library* (VCL), che include oggetti che, oltre alle API di Windows, incapsulano molte altre utili tecniche di programmazione (Windows).
- la *Borland Component Library for Cross-Platform* (CLX), che include oggetti che incapsulano la libreria Qt (Windows o Linux).

Questo capitolo descrive in breve l'ambiente di sviluppo di C++Builder e come esso si adatta perfettamente al ciclo di sviluppo di un progetto. Nella parte rimanente di questo manuale vengono forniti dettagli tecnici sullo sviluppo di applicazioni generiche, di applicazioni database, di applicazioni Internet e Intranet, oltre che sulla scrittura di controlli ActiveX e COM e di componenti personali.

Ambiente di sviluppo integrato

Quando si avvia C++Builder, ci si trova immediatamente all'interno dell'ambiente di sviluppo integrato, detto anche IDE. Questo ambiente fornisce tutti gli strumenti necessari per progettare, sviluppare, collaudare, correggere e distribuire le applicazioni, consentendo così la creazione rapida di prototipi e l'abbreviazione dei tempi del ciclo di sviluppo.

L’IDE include tutti gli strumenti necessari per iniziare la progettazione di applicazioni, quali:

- Form Designer, una finestra vuota, detta anche *scheda*, in cui progettare l’interfaccia utente (UI) dell’applicazione.
- Component palette per la visualizzazione di componenti visuali e non, che è possibile utilizzare per progettare l’interfaccia utente.
- Object Inspector per esaminare e modificare le proprietà e gli eventi degli oggetti.
- Object TreeView per visualizzare e modificare le relazioni di un componente.
- Code editor per la stesura e la modifca della logica sottostante di un programma.
- Project Manager per gestire i file che costituiscono uno o più progetti.
- Debugger integrato per ricercare e corregere gli errori nel codice.
- Molti altri strumenti, come gli editor di proprietà per modificare i valori delle properietà di un oggetto.
- Strumenti per la riga comandi che comprendono compilatori, linker e altri programmi di utilità.
- Complete librerie di classi con moltissimi oggetti riutilizzabili. La maggior parte degli oggetti forniti nella libreria di classi sono accessibili nell’IDE dalla Component palette. Per convenzione, i nomi degli oggetti nella libreria di classi iniziano con una T, come *TStatusBar*.

Alcuni strumenti potrebbero non essere inclusi in tutte le edizioni del prodotto.

Nel manuale *Per iniziare*, incluso nel prodotto, viene presentata una panoramica più completa sull’ambiente di sviluppo. Inoltre, il sistema di Guida in linea contiene la descrizione completa dei menu, delle finestre di dialogo e delle finestre dell’applicazione.

Progettazione di applicazioni

C++Builder può essere usato per progettare qualsiasi tipo di applicazione a 32 bit—dai programmi di utilità per scopi generici ai sofisticati programmi di accesso ai dati o applicazioni distribuite.

Quando si progetta in modo visuale l’interfaccia utente per l’applicazione, C++Builder genera il codice C++ sottostante. Quando si selezionano e si modificano le proprietà di componenti e schede, i risultati di queste modifiche vengono visualizzati automaticamente nel codice sorgente e viceversa. È possibile modificare i file sorgente direttamente con qualsiasi editor di testo, compreso il Code editor integrato. Le modifiche apportate vengono immediatamente riportate nell’ambiente visuale.

In C++Builder è possibile creare dei propri componenti. La maggior parte dei componenti forniti è scritta in Object Pascal. È possibile aggiungere alla Component palette i propri componenti oppure personalizzarla in base alle proprie preferenze inserendo, ad esempio, nuove pagine.

Utilizzando la libreria CLX è possibile anche utilizzare C++Builder per lo sviluppo di applicazioni eseguibili su entrambe le piattaforme Linux e Windows. CLX contiene un insieme di classi che, se utilizzate al posto di quelle incluse nella VCL, permettono di eseguire il porting del programma tra Windows e Linux. Per i dettagli sulla

programmazione per più piattaforme e per le differenze fra gli ambienti Windows e Linux, fare riferimento al Capitolo 14, "Sviluppo di applicazioni multipiattaforma".

Il Capitolo 7, "Creazione di applicazioni, componenti e librerie", presenta i supporti forniti da C++Builder per i diversi tipi di applicazioni.

Creazione di progetti

In C++Builder, tutte le attività relative allo sviluppo di applicazioni ruotano intorno ai progetti. Quando si crea un'applicazione in C++Builder si crea un progetto. Un progetto è una raccolta di file che compongono un'applicazione. Alcuni di questi file sono creati in fase di progettazione. Gli altri vengono generati automaticamente quando si compila il codice sorgente del progetto.

È possibile visualizzare il contenuto di un progetto in uno strumento per la gestione dei progetti di nome Project Manager. Il Project Manager elenca, in una vista gerarchica, i nomi delle unit, le schede contenute nella unit (nel caso ve ne siano) e mostra i percorsi dei file nel progetto. Benché sia possibile modificare direttamente la maggior parte di questi file, spesso è più semplice e affidabile utilizzare gli strumenti visuali di C++Builder.

In cima alla gerarchia del progetto, vi è un file di gruppo. È possibile combinare più progetti in un gruppo di progetti.Ciò permette di aprire nel Project Manager più di un progetto per volta. I gruppi di progetti permettono organizzare e operare su progetti correlati, come le applicazioni che funzione in combinazione fra di loro o che fanno parte di un'applicazione multi-tier. Se si lavora su un solo progetto, per creare un'applicazione non è necessario un file di gruppo di progetti.

I file di progetto, che descrivono i singoli progetti, i file e le opzioni associate, hanno l'estensione .bpr. I file di progetto contengono indicazioni per la generazione di un'applicazione o di un oggetto condiviso. Quando si aggiungono e rimuovono file utilizzando il Project Manager, il file di progetto viene aggiornato. Le opzioni di progetto vengono specificate utilizzando una finestra di dialogo, Project Options, contenente varie pagine per i diversi aspetti del progetto come le schede, l'applicazione o il compilatore. Queste opzioni di progetto vengono memorizzate nel file di progetto assieme al progetto.

Le unit e le schede sono i componenti di base di un'applicazione in C++Builder. Un progetto può condividere tutti i file form e unit esistenti, inclusi quelli che risiedono esternamente all'albero delle directory di progetto. Questo include le procedure e le funzioni custom che sono state scritte come routine indipendenti.

Se si aggiunge a un progetto un file condiviso, è bene rendersi conto che il file non viene copiato nella directory del progetto corrente; rimane nella sua ubicazione attuale. L'aggiunta al progetto corrente di un file condiviso registra il nome del file e il percorso nel file di progetto. C++Builder gestisce automaticamente questa operazione non appena si aggiungono delle unit a un progetto.

Quando si compila un progetto, non è importate la posizione in cui risiedono i file che compongono il progetto. Il compilatore tratta i file condivisi esattamente come quelli creati dal progetto stesso.

Editing del codice

Il Code editor di C++Builder è un editor ASCII completo di ogni funzione. Se si utilizza l'ambiente di programmazione visuale, viene visualizzata automaticamente una scheda come parte di un nuovo progetto. È possibile cominciare a progettare l'interfaccia dell'applicazione collocando oggetti sulla scheda e modificandone il funzionamento con l'Object Inspector. Altre attività di programmazione, invece, come la scrittura dei gestori di evento degli oggetti, devono essere fatte immettendo il codice.

I contenuti della scheda, tutte le sue proprietà, i suoi componenti e le rispettive proprietà possono essere visualizzati e modificati sotto forma di testo nel Code Editor. Nel Code Editor è possibile adattare il codice generato e aggiungervi altri componenti dall'interno dell'editor scrivendo il codice opportuno. Man mano che si scrive il codice nell'editor, il compilatore scandisce costantemente ogni modifica e aggiorna la scheda con il nuovo layout. È possibile quindi ritornare alla scheda, visualizzarla e verificare le modifiche apportate nell'editor, e quindi continuarne la messa a punto.

I sistemi di generazione del codice e di esposizione delle proprietà di C++Builder possono essere ispezionati completamente. Il codice sorgente per tutto ciò che è incluso nel file eseguibile finale—vale a dire tutti gli oggetti VCL, tutti gli oggetti CLX, i sorgenti RTL e tutti i file di progetto-possono essere visualizzati e modificati nel Code editor.

Compilazione di applicazioni

Una volta terminata la progettazione sulla scheda dell'interfaccia dell'applicazione e la scrittura di tutto il codice aggiuntivo, in modo che l'applicazione faccia ciò che ci si era prefissi, è possibile compilare il progetto dall'IDE o dalla riga comandi.

Tutti i progetti hanno come destinazione un singolo file eseguibile distribuibile. È possibile visualizzare o provare l'applicazione a diversi stadi dello sviluppo compilandola, rigenerandola o mandandola in esecuzione:

- Quando si compila, vengono ricompilate solo le unit che sono state modificate rispetto all'ultima compilazione.
- Quando si rigenera l'applicazione, vengono ricompilate tutte le unit nel progetto, indipendentemente dal fatto che siano state modificate o meno rispetto all'ultima compilazione. È importante utilizzare il comando Build anche quando si sono modificate le direttive globali del compilatore, in modo da accertarsi che tutto il codice venga compilato in base alle impostazioni appropriate. È importante anche eseguire la rigenerazione del progetto una volta modificate tutte le direttive globali del compilatore per accertarsi che tutto il codice venga compilato nello stato opportuno. È anche possibile verificare la validità del codice sorgente senza tentare di compilare il progetto.

- Quando si esegue l'applicazione, l'applicazione viene compilata e quindi eseguita. Se si è modificato il codice sorgente rispetto all'ultima compilazione, il compilatore ricompila i moduli modificati e ne riesegue il link all'applicazione.

Nel caso si siano raggruppati più progetti, è possibile compilare o rigenerare in una sola volta tutti i progetti in un singolo gruppo di progetti. Dopo aver selezionato nel Project Manager il gruppo di progetti, scegliere il comando Project | Compile All Projects o Project | Build All Projects.

Non è ancora disponibile un'applicazione C++ per Linux per compilare un'applicazione CLX in Linux, ma è possibile cominciare a sviluppare da ora le proprie applicazioni con C++Builder.

Debug di applicazioni

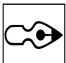
C++Builder fornisce un debugger integrato che consente di trovare e di correggere gli errori presenti nelle applicazioni. Il debugger integrato permette di controllare l'esecuzione del programma, di tenere sotto controllo i valori delle variabili e gli elementi delle strutture di dati, oltre che di modificare i valori dei dati durante la fase di debug.

Il debugger integrato può tracciare sia gli errori di esecuzione sia gli errori di logica. Eseguendo il programma fino a determinati punti ed esaminando i valori delle variabili, le funzioni nello stack delle chiamate e l'output del programma, è possibile controllare il comportamento del programma e trovare le aree in cui il comportamento non è quello progettato. Il debugger è descritto nella Guida in linea.

È anche possibile usare la gestione delle eccezioni per riconoscere, individuare e trattare gli errori. In C++Builder le eccezioni sono classi, identiche alle altre classi di C++Builder, tranne per il fatto che, per convenzione, il loro nome inizia con una E invece che con una T. Per maggiori informazioni sulla gestione delle eccezioni, consultare il Capitolo 12, "Gestione delle eccezioni".

Distribuzione di applicazioni

C++Builder include strumenti aggiuntivi che consentono la distribuzione delle applicazioni. Ad esempio, InstallShield Express (non disponibile in tutte le edizioni) aiuta a creare un package per l'installazione dell'applicazione, che include tutti i file necessari per l'esecuzione di un'applicazione distribuita. È disponibile anche il software TeamSource (non disponibile in tutte le edizioni) per tenere traccia degli aggiornamenti apportati alle applicazioni.



Non tutte le edizioni di C++Builder consentono la distribuzione di applicazioni.

Non è ancora disponibile un'applicazione C++ per Linux per distribuire un'applicazione CLX in Linux, ma è possibile cominciare a sviluppare da ora le proprie applicazioni con C++Builder.

Per informazioni specifiche sulla distribuzione, consultare il Capitolo 17, “Distribuzione di applicazioni”,.

Capitolo

3

Uso delle librerie di classi

Questo capitolo presenta una panoramica delle librerie di componenti e presenta alcuni oggetti che è possibile utilizzare durante lo sviluppo di applicazioni. C++Builder include sia la Visual Component Library (VCL) che la Borland Component Library for Cross-Platform (CLX). La VCL è per lo sviluppo in Windows e la CLX è per lo sviluppo multipiattaforma in Windows e in Linux. Sono due librerie di classi diverse ma hanno molte somiglianze.

Le librerie di classi

VCL e CLX sono librerie di classi composte da oggetti che si utilizzano durante lo sviluppo di applicazioni. Le librerie sono fra loro simili e contengono entrambe quasi tutti gli stessi oggetti. Alcuni oggetti nella VCL implementano funzioni che sono disponibili solo in Windows, come gli oggetti che vengono visualizzati sulle pagine ADO, BDE, QReport, COM+ e Server della Component palette. Potenzialmente tutti gli oggetti CLX sono disponibili sia in Windows che in Linux.

Tutti gli oggetti della VCL e della CLX discendono da *TObject*, una classe astratta i cui metodi ne definiscono il comportamento di base come la costruzione, la distruzione e la gestione dei messaggi. Quando si scrivono personalmente delle classi, dovrebbero discendere da *TObject* incluso nella libreria di classi che si intende utilizzare.

I *componenti* sono un sottoinsieme della VCL o della CLX e discendono dalla classe astratta *TComponent*. I componenti sono oggetti che possono essere collocati su una scheda o in un modulo dati e trattare in fase di progettazione. La maggior parte dei componenti sono visuali o non visuali, a seconda del fatto che siano visibili o meno in esecuzione. Alcuni componenti vengono visualizzati nella Component palette.

I componenti visuali, come *TForm* e *TSpeedButton*, sono detti controlli e discendono da *TControl*. *TControl* fornisce le proprietà che specificano gli attributi visuali dei controlli, come la loro altezza e ampiezza.

I componenti non visuali vengono usati per una vasta gamma di compiti. Ad esempio, se si sta scrivendo un'applicazione che si connete a un database, è possibile collocare sulla scheda un componente *TDataSource* per collegarsi a un controllo e un componente dataset usato dal controllo. Questa connessione non è visibile all'utente, pertanto *TDataSource* non è visuale. In progettazione, i componenti non visuali vengono rappresentati da un'icona. Ciò consente di manipolarne le proprietà e gli eventi esattamente come se si trattasse di un controllo visuale.

Durante la programmazione, è possibile accedere alla Guida in linea che contiene la documentazione di riferimento molto dettagliata su tutti gli oggetti della VCL e della CLX. Dall'interno del Code editor, collocare il cursore sull'oggetto e premere F1 per visualizzare la Guida sui componenti della topic . Gli oggetti, le proprietà, i metodi e gli eventi che sono inclusi nella VCL sono contrassegnati come "VCL Reference", mentre quelli inclusi nella CLX come "CLX Reference".

Proprietà, metodi ed eventi

Sia la VCL che la CLX sono gerarchie di oggetti legate all'IDE di Delphi, in cui è possibile sviluppare rapidamente applicazioni. Gli ogggetti in entrambe le librerie di componenti si basano su proprietà, metodi ed eventi. Ogni oggetto include i membri dati (le proprietà), le funzioni che operano sui dati (i metodi) e un modo per interagire con gli utenti della classe (gli eventi). La VCL e la CLX sono scritte in Object Pascal, anche se la VCL si basa sulle API di Windows API e la CLX si basa sulla libreria di widget Qt.

Proprietà

Le *proprietà* sono le caratteristiche di un oggetto che influiscono o sul comportamento visibile o sulle operazioni dell'oggetto. Ad esempio, la proprietà *Visible* determina se un oggetto può essere visualizzato o meno nell'interfaccia dell'applicazione. Proprietà ben progettate consentono di semplificare l'utilizzo del componente da parte di altri programmatori e la manutenzione da parte di chi l'ha progettato.

Di seguito sono riportare alcune utili caratteristiche delle proprietà:

- A differenza dei metodi, che sono disponibili solo in esecuzione, è possibile vedere e modificare le proprietà nell'IDE in fase di progettazione e ottenere un immediato riscontro visivo dei cambiamenti apportati.
- È possibile accedere alle proprietà tramite l'Object Inspector, che consente di modificare visivamente i valori dell'oggetto. L'impostazione delle proprietà in fase di progettazione è più semplice della scrittura del codice e rende il codice più semplice da gestire.
- Poiché i dati sono incapsulati, risultano protetti e privati all'interno dell'oggetto.
- Le chiamate volte a ottenere e a impostare i valori sono metodi, rendendo pertanto invisibili all'utente dell'oggetto particolari elaborazioni. Ad esempio, i dati potrebbero risiedere in una tabella, ma potrebbero apparire al programmatore come normali membri dati.

- È possibile implementare la logica che innesca eventi o modifica altri dati durante l'accesso alla proprietà. Ad esempio, la modifica del valore di una proprietà potrebbe richiedere la modifica di un'altra. È possibile modificare i metodi creati per la proprietà.
- Le proprietà possono essere virtuali.
- Una proprietà non è limitata a un singolo oggetto. La modifica di una proprietà di un oggetto potrebbe ripercuotersi su numerosi oggetti. Ad esempio, l'impostazione della proprietà *Checked* su un pulsante di opzione influenza tutti i pulsanti di opzione nel gruppo.

Metodi

Un *metodo* è una funzione che è memebro di una classe. I metodi definiscono il comportamento di un oggetto. I metodi di classe possono accedere a tutte le proprietà pubbliche, protette e *private* e ai membri dati della classe e sono comunemente menzionati come funzioni membro. Consulatre "Controllo dell'accesso" a pagina 46-4.

Eventi

Un *evento* è un'azione o un accadimento rilevato da un programma. Molte applicazioni moderne vengono definite event-driven, perché sono progettate per rispondere agli eventi. In un programma, il programmatore non ha alcun modo di prevedere la sequenza esatta di azioni che sarà eseguita dall'utente. L'utente ad esempio potrà scegliere una voce di menu, fare clic su un pulsante o selezionare del testo. Il programmatore potrà scrivere il codice per gestire gli eventi che ritiene importanti, invece di scrivere del codice che sarà eseguito sempre secondo lo stesso ordine limitato.

Indipendentemente da come è stato chiamato l'evento, C++Builder cerca di individuare se è stato predisposto un qualsiasi codice per gestire quel determinato evento. Nel caso sia stato predisposto, quel codice viene eseguito; altrimenti, ha luogo il comportamento predefinito di gestione dell'evento.

I tipi di eventi che possono verificarsi possono essere suddivisi in due categorie principali:

- Eventi utente
- Eventi di sistema

Eventi utente

Gli eventi utente sono azioni che vengono iniziate dall'utente. Esempi di eventi utente sono *OnClick* (l'utente ha fatto clic col mouse), *OnKeyPress* (l'utente ha premuto un tasto sulla tastiera), e *OnDblClick* (l'utente ha fatto doppio clic su un pulsante del mouse).

Eventi di sistema

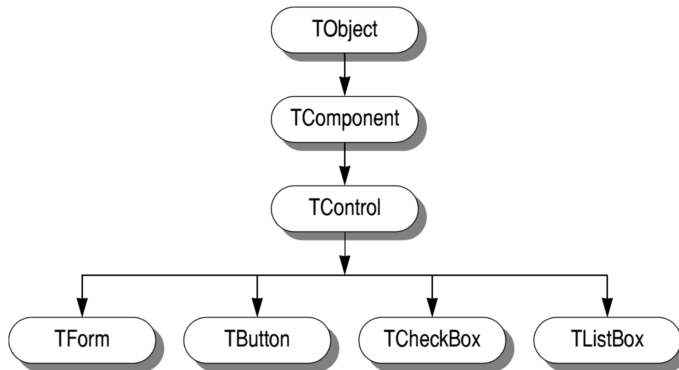
Gli eventi di sistema sono eventi generati automaticamente dal sistema operativo. Ad esempio, l'evento *OnTimer* (il componente *TTimer* genera uno di questi eventi ogni

volta che è trascorso un intervallo temporale predefinito), l'evento *OnCreate* (il componente sta per essere creato), l'evento *OnPaint* (un componente o una window devono essere ridisegnati), e così via. Di solito, gli eventi di sistema non vengono avviati direttamente da un'azione dell'utente.

Oggetti, componenti e controlli

La Figura 3.1 mostra in modo molto semplificato la gerarchia dell'ereditarietà che illustra le relazioni esistenti tra oggetti, componenti e controlli.

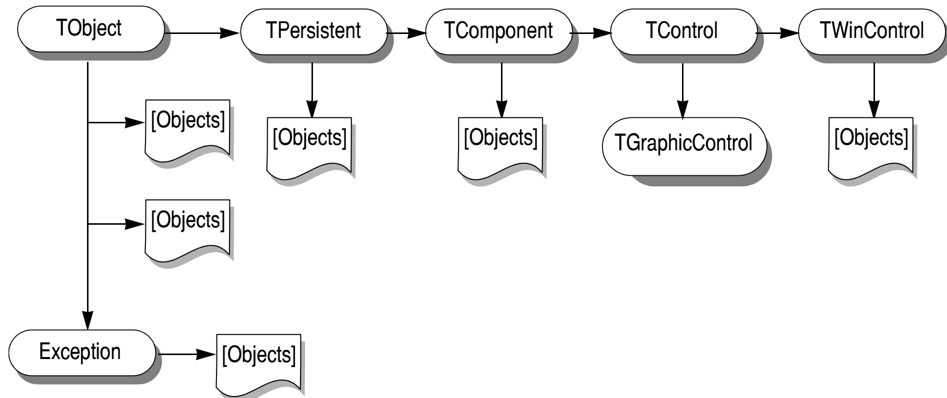
Figura 3.1 Oggetti, componenti e controlli



Ogni oggetto eredita da *TObject*, e molti oggetti ereditano da *TComponent*. I controlli, che ereditano da *TControl*, hanno la capacità di visualizzare sé stessi in fase di esecuzione. Un controllo come *TCheckBox* eredita tutta le funzionalità di *TObject*, di *TComponent* e di *TControl* e aggiunge proprie capacità specializzate.

La Figura 3.2 è una panoramica della Visual Component Library (VCL) che mostra i rami principali dell'albero dell'ereditarietà. A questo livello, la Borland Component Library for Cross-Platform (CLX) è molto simile, ma *TWinControl* è sostituito da *TWidgetControl*.

Figura 3.2 Un diagramma semplificato della gerarchia



Nella figura vengono visualizzate diverse importanti classi base che sono descritte nella tabella seguente:

Tabella 3.1 Classi di base importanti

Classe	Descrizione
TObject	Indica la classe di base e l'antenato più remoto di tutto ciò che è incluso nella VCL o nella CLX. *TObject* incapsula il comportamento fondamentale comune a tutti gli oggetti della VCL/CLX introducendo i metodi che eseguono le funzioni di base come la creazione, la gestione e la distruzione di un'istanza di un oggetto.
Exception	Specifica la classe di base di tutte le classi correlate alle eccezioni. La classe *Exception* fornisce un'interfaccia coerente per le condizioni di errore e permette alle applicazioni di gestire le condizioni di errore in modo elegante.
TPersistent	Specifica la classe di base per tutti gli oggetti che implementano proprietà. Le classi sotto *TPersistent* si occupano dell'invio di dati agli stream e permettono l'assegnazione di classi.
TComponent	Specifica la classe di base per tutti i componenti non visuali come *TApplication*. *TComponent* è l'antenato comune di tutti i componenti. Questa classe permette a un componente di essere visualizzato sulla Component palette, consente a un componente di possedere altri componenti, e consente di trattare il componente direttamente su una scheda.
TControl	Rappresenta la classe di base per tutti i controlli visibili in fase di esecuzione. *TControl* è l'antenato comune di tutti i componenti visuali e fornisce ai controlli proprietà standard come posizione e cursore. Questa classe fornisce anche gli eventi che rispondono alle azioni del mouse.
TWinControl	Specifica la classe di base di tutti gli oggetti dell'interfaccia utente. I controlli sotto *TWinControl* sono controlli basati su finestre in grado di catturare l'input da tastiera. In CLX, questi vengono chiamati widget, e *TWidgetControl* sostituisce *TWinControl*.

Le prossime sezioni presentano una descrizione generale dei tipi di classi conytenuti in ogni ramo. Per una panoramica completa della gerarchia degli oggetti della VCL e della CLX, fare riferimento ai diagramma VCL Object Hierarchy e CLX Object Hierarchy inclusi nel prodotto.

Il ramo TObject

Il ramo *TObject* include tutti gli oggetti VCL e CLX che discendono da *TObject* ma non da *TPersistent*. Molte potenti capacità degli oggetti della VCL e della CLX derivano dai metodi introdotti da *TObject*. *TObject* incapsula il comportamento di base comune a tutti gli oggetti della VCL e della CLX, introducendo metodi che forniscono:

- La capacità di rispondere quando gli oggetti vengono creati o distrutti.
- Tipo della classe e informazioni sull'istanza relativamente a un oggetto, e informazioni di tipo in esecuzione (RTTI) relativamente alle proprietà rese pubbliche.
- Supporto per la gestione di messaggi (VCL) oppure per gli eventi di sistema (CLX).

TObject è l'antenato diretto di molte classi semplici. Le classi all'interno del ramo *TObject* hanno un'importante caratteristica comune: sono transitorie. Ciò significa che queste classi non hanno un metodo per salvare lo stato in cui sono prima di essere distrutte, vale a dire che non sono persistenti.

Uno dei gruppi principali delle classi in questo ramo è la classe *Exception*. Questa classe fornisce una vasta serie di classi di eccezioni predefinite per gestire automaticamente errori di divisione per zero, errori di I/O su file, conversioni di tipo non valide e molte altre condizioni di eccezione.

Un altro tipo di classe nel ramo *TObject* sono le classi che incapsulano strutture dati, come:

- *TBits*, una classe che memorizza un "array" di valori booleani.
- *TList*, una classe di lista collegata.
- *TStack*, una classe che mantiene un array di puntatori di tipo last-in-first-out.
- *TQueue*, una classe che mantiene un array di puntatori di tipo first-in first-out.

Nella VCL è possibile anche trovare dei wrapper per oggetti esterni come *TPrinter*, che incapsula l'interfaccia per la stampante Windows e *TRegistry*, un wrapper a basso livello per il file di registro di sistema e per le funzioni che operano sul file di registro. Queste sono specifiche per l'ambiente Windows.

TStream è un buon esempio di un altro tipo di classi in questo ramo. *TStream* è il tipo di classe base per oggetti stream che sono in grado di leggere o scrivere su vari tipi di supporto, come file su disco, memoria dinamica, e così via.

Pertanto, questo ramo include molti tipi di classi differenti che risultano molto utili agli sviluppatori.

Il ramo TPersistent

Il ramo *TPersistent* include tutti gli oggetti VCL e CLX che discendono da *TPersistent* ma non da *TComponent*. La persistenza determina che cosa viene salvato con un file form o con un modulo dati e che cosa viene caricato nella scheda o nel modulo dati quando li si ricupera dalla memoria.

Gli oggetti in questo ramo implementano le proprietà per i componenti. Le proprietà vengono caricate e salvate con una scheda solo se hanno un possessore. Il possessore deve essere qualche componente. Questo ramo introduce la funzione *GetOwner* che permette di determinare il possessore della proprietà.

Gli oggetti in questo ramo sono anche i primi a includere una sezione published in cui è possibile caricare e salvare automaticamente le proprietà. Il metodo *DefineProperties* permette anche di indicare come caricare e salvare le proprietà.

Di seguito sono elencate alcune altre classi incluse nel ramo *TPersistent* della gerarchia:

- *TGraphicsObject,* una classe base astratta per oggetti grafici, come *TBrush*, *TFont*, e *TPen*.
- *TGraphic,* una classe base astratta per oggetti come *TBitmap* e *TIcon* ,in grado di memorizzare e visualizzare immagini.
- *TStrings,* una classe base per oggetti che rappresentano un elenco di stringhe.
- *TClipboard,* una classe in grado di contenere testo o immagini grafiche tagliate o copiate da un'applicazione.
- *TCollection*, *TOwnedCollection*, e *TCollectionItem,* classi che contengono raccolte indicizzate di elementi definiti in modo particolare.

Il ramo TComponent

Il ramo *TComponent* contiene oggetti che scendono da *TComponent* ma non da *TControl*. Gli oggetti in questo ramo sono componenti che è possibile manipolare sulle schede in fase di progettazione. Essi sono oggetti persistenti che possono fare quanto segue:

- Essere visualizzati sulla Component palette e modificati nel Form designer.
- Possedere e gestire altri componenti.
- Caricare e salvare se stessi.

Diversi metodi in *TComponent* regolano il comportamento dei componenti in fase di progettazione e quali informazioni vengono salvate con il componente. In questo ramo della CLX viene introdotto il concetto di stream. C gestisce automaticamente la maggior parte delle operazioni che coinvolgono gli stream. Le proprietà sono persistenti se sono state rese pubbliche e le proprietà rese pubbliche sono automaticamente soggette allo stream.

La classe *TComponent* introduce anche il concetto di possesso che è propagato in tutta la VCL e la CLX. Due proprietà supportano il possesso: *Owner* e *Components*. Ogni componente ha una proprietà *Owner* che fa riferimento a un altro componente come possessore. Un componente può possedere altri componenti. In questo caso, tutti i componenti posseduti sono referenziati nella proprietà *Array* del componente.

Il costruttore di un componente richiede un singolo parametro che è utilizzato per specificare il possessore del nuovo componente. Se il possessore che viene passato esiste, il nuovo componente viene aggiunto alla lista *Components* del possessore. Oltre all'utilizzo della lista *Components* per referenziare i componenti posseduti, questa proprietà consente anche la distruzione automatica dei componenti posseduti. Finché

il componente ha un possessore, sarà distrutto al momento della distruzione del possessore. Ad esempio, poiché *TForm* è un discendente di *TComponent*, quando si distruggerà la scheda saranno distrutti tutti i componenti posseduti dalla scheda e la memoria allocata sarà liberata. In questo caso si parte dall'assunto che tutti i componenti sulla scheda siano in grado di far pulizia di sé stessi correttamente nel momento in cui vengono chiamati i rispettivi distruttori.

Se un tipo di proprietà è un *TComponent* o un discendente, il sistema di streaming crea un'istanza di quel tipo al momento della sua lettura. Se un tipo di proprietà è *TPersistent* ma non *TComponent*, il sistema di streaming utilizza l'istanza esistente disponibile attraverso la proprietà e legge i valori delle proprietà di quell'istanza.

Quando si crea un file form (un file utilizzato per memorizzare le informazioni relative ai componenti sulla scheda), il Form designer analizza ciclicamente la sua matrice di componenti e salva tutti i componenti sulla scheda. Ogni componente "sa" come scrivere nello stream (in questo caso, un file di testo) le proprietà che sono state modificate. Al contrario, quando si caricano le proprietà dei componenti nel file di scheda, il Form designer analizza ciclicamente la sua matrice di componenti e carica ogni componente.

I tipi di classi che troverai in questo ramo includono:

- *TActionList*, una classe che conserva un elenco di azioni utilizzate con componenti e controlli, come elementi di menu e pulsanti.
- *TMainMenu*, una classe che fornisce alle schede una barra di menu e i relativi menu a discesa.
- *TOpenDialog*, *TSaveDialog*, *TFontDialog*, *TFindDialog*, *TColorDialog* e così via, sono classi che rendono disponibili le finestre di dialogo utilizzate più di frequente.
- *TScreen*, una classe che tiene traccia di quali schede e moduli dati sono stati istanziati dall'applicazione, qual è scheda attiva, qual è il controllo attivo all'interno di quella scheda, le dimensioni e la risoluzione dello schermo, e i cursori e i font disponibile utilizzabili dall'applicazione.

Componenti che non hanno bisogno di un'interfaccia visuale possono essere derivati direttamente da *TComponent*. Per creare uno strumento come un dispositivo *TTimer*, è possibile derivarlo da *TComponent*. Questo tipo di componente risiede nella Component palette, ma esegue funzioni interne a cui si accede mediante codice e non viene visualizzato nell'interfaccia utente in fase di esecuzione.

Nella CLX, il ramo *TComponent* include anche *THandleComponent*. Questa è la classe di base per i componenti non visuali che richiedono un handle per un oggetto Qt sottostante, come le finestre di dialogo e i menu.

Per i dettagli sull'impostazione delle proprietà, sulla chiamata ai metodi e sulle operazioni con gli eventi dei componenti, consultare il Capitolo 5, "Operazioni con componenti".

Il ramo TControl

Il ramo *TControl* è formato da componenti che discendono da *TControl* ma non da *TWinControl* (*TWidgetControl* nella CLX). Gli oggetti in questo ramo sono controlli che sono oggetti visuali che l'utente dll'applicazione può vedere e manipolare in

esecuzione. Tutti i controlli hanno proprietà, metodi ed eventi in comune che sono correlati all'aspetto del controllo, come la posizione, il cursore associato alla finestra del controllo (al widget nella CLX), i metodi per colorare o spostare il controllo e gli eventi per rispondere alle azioni del mouse. I controlli non possono mai ricevere l'input da tastiera.

Mentre *TComponent* definisce il comportamento di tutti i componenti, *TControl* definisce il comportamento per tutti i controlli visuali. Ciò comprende le routine di disegno, gli eventi standard e le relazioni contenuto-contenitore.

Tutti i controlli visuali condividono alcune proprietà. Benché queste proprietà siano ereditate da *TControl*, esse vengo rese pubbliche, e appaiono pertanto nell'Object Inspector, solo per quei componenti per cui sono applicabili. Ad esempio, il componente *TImage* non pubblica la proprietà *Color*, poiché il suo colore è determinato dall'immagine grafica visualizzata.

Ci sono due tipi di controlli:

- Quelli che hanno una propria finestra (o un widget.
- Quelli che utilizzano la finestra (o il widget) del proprio genitore.

I controlli che hanno una propria finestra sono detti controlli "con finestra" (VCL) o controlli "basati su widget" (CLX) e discendono da *TWinControl* (*TWidgetControl* in CLX). I pulsanti e le caselle di controllo rientrano in questa categoria.

I controlli che utilizzano la finestra (o il widget) del genitore sono definiti controlli grafici e discendono da *TGraphicControl*. I controlli immagine ed etichetta rientrano in questa categoria. I controlli grafici non posseggono un handle e non possono ricevere il fuoco. Poiché un controllo grafico non necessita di un handle, utilizza meno risorse di sistema. I controlli grafici devono tracciare sé stessi e non possono essere genitori di altri controlli.

Per informazioni sugli altri controlli grafici, vedere "Controlli grafici" a pagina 9-19, e per i dettagli sui diversi tipi di controlli consultare il Capitolo 9, "Tipi di controlli". Per informazioni su come interagire con i controlli in esecuzione, consultare il Capitolo 6, "Uso dei controlli".

Il ramo TWinControl/TWidgetControl

Nella VCL, il ramo *TWinControl* include tutti i controlli che discendono da *TWinControl*. *TWinControl* è la classe di base per tutti i controlli con finestra, che sono elementi che si utilizzeranno nell'interfaccia utente di un'applicazione. I controlli con finestra sono wrapper di controlli Windows.

In CLX, *TWidgetControl*, che sostituisce *TWinControl*, è la classe base per tutti i controlli widget, che sono wrappers di *widget*.

I controlli con finestra e i widget:

- Possono ricevere il fuoco durante la fase di esecuzione di un'applicazione, il che significa che possono ricevere l'input da tastiera da parte dell'utente dell'applicazione. A confronto, altri controlli possono solo visualizzare dati.
- Possono essere genitori di uno o più controlli figlio.

- Hanno un handle, o un identificatore univoco.

Il ramo *TWinControl*/TwidgetControl include sia i controlli che vengono tracciati automaticamente (inclusi *TEdit*, *TListBox*, *TComboBox*, *TPageControl*, e così via) sia i controlli custom che devono essere tracciati da C++Builder come *TDBNavigator*, *TMediaPlayer* (solo nella VCL), *TGauge* (solo nella VCL). Di solito i discendenti diretti di *TWinControl*/*TWidgetControl* implementano controlli standard, come i campi di editing, le caselle combinate, le caselle di riepilogo o i controlli pagina e, pertanto, sanno già come tracciarsi.

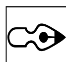
La classe *TCustomControl* viene fornita per quei componenti che richiedono un handle di finestra ma non incapsulano un controllo standard che include la capacità di tracciarsi. Non è necessario preoccuparsi di come i controlli vengono tracciati o di come rispondono agli eventi. C++Buider incapsula completamente e automaticamnete questo comportamento.

Capitolo

4

Utilizzo di BaseCLX

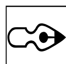
Vi sono molte unit comuni sia alla VCL che alla CLX che forniscono il supporto sottostante per entrambe le librerie di componenti. Collettivamente, queste unit vengono chiamate BaseCLX. BaseCLX non include nessuno dei componenti visualizzati sulla Component palette. Include invece moltissime classi e routine globali che vengono usate dai componenti che vengono visualizzati sulla Component palette. Queste classi e queste routine sono utilizzabili anche nel codice delle proprie applicazioni o durante la scrittura delle proprie classi.



Le routine globali che compongono la BaseCLX sono anche dette libreria di esecuzione. Non confondere queste routine con la libreria di esecuzione del C++. Molte di queste eseguono funzioni simili a quelle nella libreria di esecuzione del C++, ma possono essere distinte in quanto i nomi delle funzioni iniziano con una lettera maiuscola e vengono dichiarate nello header di una unit.

Le sezioni seguenti trattano molte classi e routine che compongono BaseCLX e ne illustrano i possibili utilizzi. Fra questi vi sono:

- Uso di stream
- Operazioni con i file
- Operazioni con i file .ini
- Operazioni con le liste
- Operazioni con elenchi di stringhe
- Operazioni con stringhe
- Conversione di misure
- Creazione di spazi di disegno



Questo elenco di attività non è esaustivo. La libreria di esecuzione in BaseCLX contiene molte routine in grado di eseguire compiti non citati in questa trattazione. Fra qyesti vi sono una serie di funzioni matematiche (definite nella unit Math), varie routine per operare con valori di data/ora (definiti nelle unit SysUtils e DateUtils) e routine per operare con i Variant di Object Pascal (definite nella unit Variants).

Uso di stream

Gli stream sono classi che permettono di leggere e scrivere dati. Essi forniscono un'interfaccia comune per la lettura e la scrittura su svariati supporti come memoria, stringhe, socket e campi BLOB nei database. Esistono numerose classi stream, che discendono tutte da *TStream*. Ogni classe stream è specifica per un tipo di supporto. Ad esempio, *TMemoryStream* legge o scrive su un'immagine in memoria, *TFileStream* legge o scrive su un file.

Uso di stream per leggere o scrivere dati

Le classi stream condividono tutte svariati metodi per la lettura e la scrittura di dati. Questi metodi sono distinguibili per il fatto che compiono quanto segue:

- Restituiscono il numero di byte letti o scritti.
- Richiedono che sia noto il numero di byte.
- Generano un'eccezione sull'errore.

Metodi di stream per lettura e scrittura

Il metodo *Read* legge dallo stream un numero specificato di byte, iniziando dalla sua posizione corrente (*Position*), caricandoli in un buffer. La lettura quindi sposta la posizione corrente del numero di byte effettivamente trasferiti. Il prototipo per *Read* è

```
virtual int __fastcall Read(void *Buffer, int Count);
```

Read è utile quando il numero di byte del file non è noto. *Read* restituisce il numero di byte effettivamente trasferiti, che può essere minore di *Count* se lo stream non conteneva *Count* byte di dati precedenti alla posizione corrente.

Write è una funzione che scrive *Count* byte dal buffer nel file associato allo stream, partendo dalla *Position* corrente. Il prototipo per *Write* è:

```
virtual int __fastcall Write(const void *Buffer, int Count);
```

Dopo la scrittura nel file, *Write* sposta la posizione corrente del numero di byte scritti, e restituisce il numero di byte effettivamente scritti, che può essere minore di *Count* se incontra la fine del buffer oppure se lo stream non può accettare più byte.

Le procedure complementari sono *ReadBuffer* e *WriteBuffer* che, a differenza di *Read* e *Write*, non restituiscono il numero di byte letti o scritti. Queste procedure sono utili nei casi in cui il numero di byte è noto e sono necessarie, per esempio, per la lettura delle strutture. *ReadBuffer* e *WriteBuffer* generano un'eccezione (*EReadError* e *EWriteError*) se il *Count* byte non corrisponde esattamente. Ciò è in contrasto con i metodi *Read* e *Write*, che possono restituire un numero di byte diverso dal valore chiesto. I prototipi per *ReadBuffer* e *WriteBuffer* sono:

```
virtual int __fastcall ReadBuffer(void *Buffer, int Count);

virtual int __fastcall WriteBuffer(const void *Buffer, int Count);
```

Questi metodi chiamano i metodi *Read* e *Write*, per eseguire una lettura e una scrittura effettive.

Lettura e scrittura di componenti

TStream definisce metodi specializzati, *ReadComponent* e *WriteComponent*, per la lettura e la scrittura di componenti. È possibile utilizzarli nelle applicazioni come un modo per salvare i componenti e le rispettive proprietà quando li si crea o li si modifica in fase di esecuzione.

ReadComponent e *WriteComponent* sono i metodi che l'IDE utilizza per leggere o scrivere i componenti nei file scheda. Durante la lettura o la scrittura da o in un file scheda, le classi stream operano con le classi *TFiler*, *TReader* e *TWriter*, per leggere oggetti dal file scheda o per scriverli su disco. Per ulteriori informazioni sull'uso del sistema di streaming dei componenti, consultare nella guida di riferimento in linea le classi *TStream*, *TFiler*, *TReader*, *TWriter*, e *TComponent*.

Copia di dati da uno stream a un altro

Quando si copiano dati da uno stream a un altro, non è necessario leggere esplicitamente e poi scrivere i dati. È possibile invece utilizzare il metodo *CopyFrom*, come illustrato nell'esempio seguente.

L'applicazione include due controlli di testo (From e To) e un pulsante Copy File.

```
void __fastcall TForm1::CopyFileClick(TObject *Sender)
{
  TStream* stream1= new TFileStream(From->Text,fmOpenRead | fmShareDenyWrite);
  try
  {
    TStream* stream2 = new TFileStream(To->Text, fmOpenWrite | fmShareDenyRead);
    try
    {
      stream2 -> CopyFrom(stream1, stream1->Size);
    }
    __finally
    {
      delete stream2;
    }
  }
  __finally
  {
    delete stream1;
  }
}
```

Specifica della posizione e della dimensione dello stream

Oltre ai metodi per leggere e scrivere, questi oggetti consentono alle applicazioni di posizionarsi ad una posizione arbitraria nello stream oppure di cambiarne le dimensioni . Una volta individuata la posizione specificata, la successiva operazione di lettura o di scrittura inizia la lettura da o la scrittura nello stream a partire da quella posizione.

Ricerca di una posizione specifica

Il metodo *Seek* è il meccanismo più generale per spostarsi a una particolare posizione nello stream. Esistono due ridefinizioni per il metodo *Seek*:

```
virtual int __fastcall  Seek(int Offset, Word Origin);
virtual __int64 __fastcall  Seek(const __int64 Offset, TSeekOrigin Origin);
```

Entrambe le definizoni funzionano allo stesso modo. La differenza è che una versione utilizza un intero a 32 bit per rappresentare le posizioni e gli offset, mentre l'altro utilizza un intero a 64 bit.

Il parametro *Origin* indica come interpretare il parametro *Offset*. *Origin* sarà uno dei seguenti valori:

Valore	Significato
soFromBeginning	Offset è dall'inizio della risorsa. *Seek* sposta alla posizione Offset. Offset deve essere >= 0.
soFromCurrent	Offset è dalla posizione corrente nella risorsa. *Seek* sposta alla *Position* È Offset.
soFromEnd	Offset è dalla fine della risorsa. Offset deve essere <= 0 per indicare un numero di byte prima della fine del file.

Seek azzera la *Position* corrente dello stream, spostandola dello scostamento indicato. *Seek* restituisce la nuova posizione corrente nello stream.

Utilizzo delle proprietà Position e Size

Tutti gli stream hanno una serie di proprietà che contengono la posizione e le dimensioni correnti. Queste proprietà sono utilizzate dal metodo *Seek*, oltre che da tutti i metodi che leggono dallo o scrivono nello stream.

La proprietà *Position* indica l'offset corrente, in byte, nello stream (a partire dall'inizio dello stream di dati).

La proprietà *Size* indica le dimensioni in byte dello stream. Può essere utilizzata per determinare il numero di byte disponibili per la lettura o per troncare i dati nello stream.

Size viene usata internamente dalle routine che leggono e scrivono nello stream.

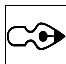
L'impostazione della proprietà *Size* cambia le dimensioni dei dati nello stream. Ad esempio,in un file stream, viene usata come marcatore di fine file per troncare il file. Se le dimensioni *Size* del stream non possono essere cambiate, viene sollevata un'eccezione. Ad esempio, il tentativo di cambiare *Size* di uno stream di file di sola lettura solleva un'eccezione.

Operazioni con i file

BaseCLX supporta diversi modi per operare con i file. Oltre all'utilizzo degli stream di file, ci sono numerose routine di libreria di esecuzione per operazioni di I/O si file.

Sia gli stream di file che le routine globali per leggere da e scrivere su file sono descritti in “Approccio alle operazioni di I/O su file” on page 4-5.

Oltre alle operazioni di input/output, si potrebbero volere trattare i file su disco. Il supporto per le operazioni sui file invece che sul loro contenuto è descritto in “Manipolazione dei file” a pagina 4-7.



Durante l’utilizzo della CLX in applicazioni cross-platform, è bene ricordare che benché il linguaggio Object Pascal non sia sensibile alle differenze fra maiuscole e minuscole, lo è invece il sistema operativo Linux. Utilizzando gli oggetti e le routine che operano con i file, bisogna prestare attenzione a come sono scritti i nomi dei file.

Approccio alle operazioni di I/O su file

Ci sono tre modalità per affrontare le operazioni di lettura e di scrittura su file:

- Il metodo consigliato per operare con i file è l’utilizzo degli stream di file. Gli stream di file sono istanze oggetto della classe *TFileStream* utilizzata per accedere alle informazioni in file su disco. Gli stream di file sono un approccio portatile e di alto livello alle operazoni di I/O su file. Poiché gli stream di file rendono disponibile l’handle del file, questo approccio può essere combinato con quello successivo. La sezione successiva, “Uso degli stream di file” tratta in dettaglio *TFileStream*.
- È possibile operare con file utilizzando un approccio basato su handle. Gli handle di file sono forniti dal sistema operativo quando si crea un file o lo si apre per operare con il suo contenuto. La unit SysUtils definisce molte routine di gestione file che operano con file utilizzando un handle di file. In Windows, di solito, queste routine sono wrapper per funzioni API di Windows. Poiché le funzioni Delphi usano la sintassi Object Pascal e occasionalmente forniscono valori di parametri predefiniti, esse costituiscono una comoda interfaccia per le API di Windows. Inoltre, esistono le versioni corrispondenti in Linux, cosicché è possibile utilizzare queste routine in applicazioni multipiattaforma. Per utilizzare un approccio basato su handle, per prima cosa si deve aprire apre un file utilizzando la funzione *FileOpen* o creare un nuovo file utilizzando la funzione *FileCreate*. Una volta ottenuto l’handle, utilizzare routine basate su handle per operare con il contenuto (scrittura di una riga, lettura del testo, e così via).
- La libreria di esecuzione del C e la libreria standard C++ includono molte funzioni e classi per le attività con file. Queste hanno il vantaggio che possono essere utilizzate in applicazioni che non utilizzano la VCL o le CLX. Per informazioni su queste funzioni, vedere la documentazione online per la libreria di esecuzione del C o per la libreria standard C++.

Uso degli stream di file

TFileStream è una classe che consente alle applicazioni di effettuare operazioni di lettura e scrittura sui file su disco. È usata per le rappresentazioni ad alto livello degli oggetti degli stream di file. Poiché *TFileStream* è un oggetto stream, condivide i metodi dei comuni stream. È possibile utilizzare questi metodi per leggere da o per

scrivere su un file, per copiare i dati da altre classi stream o in altre classi, e leggere o scrivere i valori dei componenti. Vedere "Uso di stream" a pagina 4-2 per i dettagli sulle capacita che gli stream di file ereditano essendo classi stream.

Inoltre, gli stream di file danno accesso allo handle dei file, rendendo così possibile il loro utilizzo con routine globali di gestione dei file che richiedono l'handle di file.

Creazione e apertura di file con stream di file

Per creare o aprire un file e avere accesso ad un handle per il file, basta istanziare un *TFileStream*. Questa operazione apre un determinato file e fornisce i metodi per la lettura o la scrittura relative. Se il file non può essere aperto, *TFileStream* solleva un'eccezione.

```
__fastcall TFileStream(const AnsiString FileName, Word Mode);
```

Il parametro *Mode* specifica come verrà aperto il file quando si crea lo stream di file. Il parametro *Mode* è costituito da una modalità open e da una modalità share OR+ed contemporaneamente. La modalità open può assumere uno dei seguenti valori:

Tabella 4.1 Modalità Open

Valore	Significato
fmCreate	Inserisce in *TFileStream* un file con un determinato nome. Se esiste un file con quel dato nome, apre il file in modo scrittura.
fmOpenRead	Apre il file per la sola lettura.
fmOpenWrite	Apre il file per la sola scrittura. La scrittura nel file sostituisce completamente il contenuto attivo.
fmOpenReadWrite	Apre il file per modificare il contenuto attivo anziché sostituirlo.

La modalità share può assumere uno dei seguenti valori con le limitazioni elencate di seguito:

Tabella 4.2 Modalità Share

Valore	Significato
fmShareCompat	La condivisione è compatibile con il modo in cui vengono aperti gli FCB.
fmShareExclusive	Le altre applicazioni non possono aprire il file per nessun motivo.
fmShareDenyWrite	Le altre applicazioni possono aprire il file per la lettura, ma non per la scrittura.
fmShareDenyRead	Le altre applicazioni possono aprire il file per la scrittura, ma non per la lettura.
fmShareDenyNone	Non viene fatto alcun tentativo per impedire alle applicazioni di leggere da o scrivere nel file.

Si noti che la modalità share che è possibile utilizzare dipende dalla modalità open che è stata utilizzata. La seguente tabella mostra le modalità share disponibili per ogni modalità open.

Tabella 4.3 Modalità shared disponibili per ogni modalità open

Open Mode	fmShareCompat	fmShareExclusive	fmShareDenyWrite	fmShareDenyRead	fmShareDenyNone
fmOpenRead	Non utilizzabile	Non utilizzabile	Disponibile	Non utilizzabile	Disponibile
fmOpenWrite	Disponibile	Disponibile	Non utilizzabile	Disponibile	Disponibile
fmOpenReadWrite	Disponibile	Disponibile	Disponibile	Disponibile	Disponibile

Le costanti delle modalità open e share sono definite nella unit SysUtils.

Uso dell'handle di file

Quando si istanzia *TFileStream*, si ottiene l'accesso all'handle dei file. Questo file è contenuto nella proprietà *Handle*. In Windows, *Handle* è un handle di file Windows. Nelle versioni Linux della CLX, è un handle di file Linux. *Handle* è una proprietà a sola lettura che indica il modo in cui il file viene aperto. Se si vogliono cambiare gli attributi dell'handle di file, occorre creare un nuovo oggetto flusso di file.

Alcune routine per la manipolazione dei file accettano un handle di file come parametro. Quando si ha uno stream di file, è possibile usare la proprietà *Handle* in qualsiasi situazione in cui si può usare un handle di file. Bisogna prestare attenzione al fatto che, a differenza dei flussi di handle, i flussi di file chiudono gli handle dei file quando l'oggetto viene distrutto.

Manipolazione dei file

Nella libreria runtime di the BaseCLX sono incorporate molte delle comuni operazioni sui file. Le procedure e le funzioni per lavorare con i file operano ad alto livello. Per la maggior parte delle routine, bisogna specificare il nome del file e la routine che effettua automaticamente le chiamate necessarie al sistema operativo. In alcuni casi, si utilizzano invece handle di file.



Benché il linguaggio Object Pascal non sia sensibile alle differenze fra maiuscole e minuscole, lo è invece il sistema operativo Linux. Prestare molta attenzione all'uso di maiuscole/minuscole quando si lavora con file in applicazioni multipiattaforma.

Cancellazione di un file

La cancellazione elimina il file dal disco e ne rimuove la voce dalla relativa directory. Non esiste una operazione corrispondente che permetta di ripristinare un file cancellato; generalmente, quindi, le applicazioni chiederanno agli utenti di confermare la cancellazione dei file. Per cancellare un file, basta passare il suo nome alla funzione *DeleteFile*:

```
DeleteFile(FileName);
```

DeleteFile restituisce **true** se il file viene cancellato, **false** in caso contrario (per esempio, se il file non esiste o se è a sola lettura). *DeleteFile* elimina il file chiamato da *FileName* dal disco.

Ricerca di un file

Per trovare un file si usano tre routine: *FindFirst*, *FindNext*, and *FindClose*. *FindFirst* cerca la prima istanza del nome di un file con un certo set di attributi in una determinata directory. *FindNext* restituisce il primo file corrispondente al nome e agli attributi specificati in una precedente chiamata a *FindFirst*. *FindClose* rilascia la memoria allocata da *FindFirst*. Si deve usare sempre *FindClose* per concludere una sequenza *FindFirst*/*FindNext*. Per sapere se un file esiste, c'è una funzione *FileExists* che restituisce **true** in caso affermativo, **false** in caso contrario.

Le tre routine di ricerca dei file accettano *TSearchRec* come parametro. *TSearchRec* definisce le informazioni sul file cercate da *FindFirst* o *FindNext*. La dichiarazione per *TSearchRec* è:

```
struct TSearchRec
{
   int Time; // time stamp of the file
   int Size; // size of the file in bytes
   int Attr; // file attribute flags
   AnsiString Name; // filename and extension
   int ExcludeAttr; // file attribute flags for files to ignore
   unsigned FindHandle;
   _WIN32_FIND_DATAA FindData; // structure with addition information
} ;
```

Se viene trovato un file, i campi del parametro di tipo *TSearchRec* vengono modificati per descrivere il file trovato. È possibile verificare *Attr* con le seguenti costanti o valori per determinare se un file ha uno specifico attributo:

Tabella 4.4 Costanti di attributo e valori

Costante	Valore	Descrizione
faReadOnly	0x00000001	File a sola lettura
faHidden	0x00000002	File nascosti
faSysFile	0x00000004	File di sistema
faVolumeID	0x00000008	File ID volume
faDirectory	0x00000010	File di directory
faArchive	0x00000020	File archivio
faAnyFile	0x0000003F	File di qualsiasi tipo

Per verificare un attributo, bisogna abbinare il valore del campo *Attr* alla costante di attributo con l'operatore **&**. Se il file ha quell'attributo, il risultato sarà maggiore di 0. Ad esempio, se il file trovato è un file nascosto, la seguente espressione sarà valutata come **true**: (*SearchRec.Attr* and *faHidden*> 0). Gli attributi possono essere combinati eseguendo l'OR delle rispettive costanti o dei valori. Ad esempio, per cercare i file nascosti e a sola lettura oltre ai normali file, si può passare l'attributo (faReadOnlyoppure *faHidden*) al parametro *Attr*.

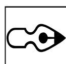


Questo esempio usa un'etichetta, un pulsante di nome *Search* e un pulsante di nome *Again* su una scheda. Quando l'utente fa clic sul pulsante *Search*, viene trovato il primo file nel percorso specificato, e il nome e il numero di byte del file appaiono nel titolo dell'etichetta.Ogni volta che l'utente fa clic sul pulsante *Again*, nell'etichetta vengono visualizzati le dimensioni e il nome corrispondenti del file:

```
  TSearchRec SearchRec; // global variable

void __fastcall TForm1::SearchClick(TObject *Sender)
{
  FindFirst("c:\\Program Files\\bcb6\\bin\\*.*", faAnyFile, SearchRec);
  Label1->Caption = SearchRec->Name + " is " + IntToStr(SearchRec.Size) + " bytes in size";
}

void __fastcall TForm1::AgainClick(TObject *Sender)
{
  if (FindNext(SearchRec) == 0)
    Label1->Caption = SearchRec->Name + " is " + IntToStr(SearchRec.Size) + " bytes in
size";
  else
    FindClose(SearchRec);
}
```



Nelle applicazioni multipiattaforma, tutti i nomi di percorso scritti direttamente nel codice vanno sostituiti con il percorso corretto per il sistema, oppure si devono utilizzare variabili di ambiente per rappresentare tali percorsi (sulla pagina Environment Variables selezionare il comando Tools | Environment Options.

Cambiamento del nome di un file

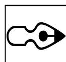
Per modificare un nome file, utilizzare la funzione *RenameFile*:

```
extern PACKAGE bool __fastcall RenameFile(const AnsiString OldName, const AnsiString
NewName);
```

RenameFile cambia il nome, identificato da *OldFileName*, con il nome specificato da *NewFileName*. Se l'operazione ha successo, *RenameFile* restituisce **true**. Se non è possibile rinominare il file, (ad esempio, se un file chiamato *NewFileName* esiste già), *RenameFile* restituisce **false**. Ad esempio:

```
if (!RenameFile("OLDNAME.TXT","NEWNAME.TXT"))
  ErrorMsg("Error renaming file!");
```

È impossibile rinominare (spostare) un file nelle unità usando *RenameFile*. Prima bisogna copiare il file e poi cancellare il vecchio.



RenameFile nella libreria runtime BaseCLX è un wrapper per la funzione *MoveFile* delle API di Windows, e pertanto nemmeno *MoveFile* funzionerà sulle varie unità.

Routine per data-ora

Le routine *FileAge*, *FileGetDate* e *FileSetDate* operano sui valori data-ora del sistema operativo. *FileAge* restituisce la data e l'ora di registrazione di un file, o -1 se il file non esiste. *FileSetDate* imposta la data e l'ora di registrazione per un determinato file, e restituisce zero se l'esito è positivo, o un codice di errore se l'esito è negativo.

FileGetDate restituisce la data e l'ora di registrazione per il file specificato o -1 se l'handle non è corretto.

Analogamente a molte altre routine per il trattamento dei file, *FileAge* utilizza un nome di file di tipo string. *FileGetDate* e *FileSetDate*, tuttavia, utilizzano un parametro integer che accetta un handle di file. Per ottenere l'handle di file è possibile

- Utilizzare la funzione *FileOpen* o *FileCreate* per creare un nuovo file o aprire un file esistente. Sia *FileOpen* che *FileCreate* restituiscono un handle di file.
- Istanziare *TFileStream* per creare o aprire un file. Quindi utilizzarne la proprietà *Handle*. Per ulteriori informazioni, vedere "Uso degli stream di file" a pagina 4-5.

Copia di un file

La libreria runtime non fornisce tutte le routine per copiare un file. Comunque, se si stanno scrivendo applicazioni per l'ambiente Windows, per chiamare un file, è possibile chiamare direttamente la funzione API *CopyFile* di Windows. Come la maggior parte delle routine della libreria runtime , *CopyFile* accetta un nome di file come parametro, e non un handle di file. Quando si copia un file, occorre fare attenzione che nel nuovo file vengano copiati gli attributi del file esistente, ma non quelli relativi alla sicurezza. *CopyFile* è utile anche nello spostamento dei file nelle unità, perché né la funzione *RenameFile* né la funzione API *MoveFile* possono rinominare oppure spostare i file nelle unità. Per ulteriori informazioni, consultare la Guida in linea di Microsoft Windows.

Operazioni con file ini e con il Registro di sistema

Molte applicazioni utilizzano i file ini per memorizzare le informazioni di configurazione. BaseCLX include due classi per le operazioni con i file ini: *TIniFile* e *TMemIniFile*. L'utilizzo dei file ini ha il vantaggio che possono essere utilizzati in applicazioni multipiattaforma e sono semplici da leggere e da modificare. Per informazioni su queste classi, vedere "Utilizzo di TIniFile e di TMemIniFile" a pagina 4-11 per ulteriori informazioni.

Molte applicazioni Windows sostituiscono l'utilizzo di file ini con il Registro di sistema. Il file di registro di sistema di Windows è un database gerarchico che agisce come uno spazio di memoria centralizzato per informazioni di configurazione . La VCL include classi per operare con il Registro di sistema. Benché dal punto di vista tecnico non facciano parte di BaseCLX (poiché essi sono disponibili solo in Windows), due di queste classi, *TRegistryIniFile* e *TRegistry,* sono trattati in questa sezione grazie alla loro somiglianza con le classi usate per operare con i file ini.

TRegistryIniFile è utile per applicazioni multipiattaforma, poiché condivide un antenato comune (*TCustomIniFile*) con le classi che operano con file ini. Se ci si limita ad utilizzare i metodi dell'antenato comune (*TCustomIniFile*) l'applicazione può funzionare su entrambe le applicazioni con pochissimo codice condizionale. *TRegistryIniFile* è trattato in "Utilizzo di TRegistryIniFile" a pagina 4-12.

Nel caso di applicazioni che non sono multipiattaforma, è possibile utilizzare la classe *TRegistry*. Le proprietà e i metodi di *TRegistry* hanno nomi che corrispondono

più direttamente al modo in cui è organizzato il Registro di sistema, in quanto non è necessario che siano compatibili con le classi per i file ini. *TRegistry* è trattato in "Uso di TRegistry" a pagina 4-12.

Utilizzo di TIniFile e di TMemIniFile

Il formato dei file ini è ancora molto diffuso, e molti file di configurazione (come il file DSK delle impostazioni del Desktop) sono in questo formato. Questo formato è particolarmente utile in applicazioni multipiattaforma, in cui è impossibile fare affidamento su un Registro di sistema per la memorizzazione delle informazioni di configurazione. BaseCLX fornisce due classi, *TIniFile* e *TMemIniFile*, per semplificare le operazioni di lettura e di scrittura con i file ini.

In Linux, *TMemIniFile* e *TIniFile* sono identiche. Su Windows, *TIniFile* opera direttamente sul file ini su disco mentre *TMemIniFile* registra tutte le modifiche in memoria e non le scrive sul disco finché non si chiama il metodo *UpdateFile*.

Quando si crea un'istanza dell'oggetto *TIniFile* oppure *TMemIniFile* , si passa al costruttore il nome del file INI come parametro. Se il file non esiste, viene creato automaticamente. Sarà quindi possibile leggere i valori utilizzando metodi di lettura come *ReadString*,*ReadDate ReadInteger* o *ReadBool*. In alternativa, se si desidera leggere un'intera sezione del file ini, è possibile utilizzare il metodo *ReadSection*. Analogamente, è possibile scrivere valori utilizzando metodi come *WriteBool*, *WriteInteger*, o *WriteString*.

Di seguito è riportato un esempio di lettura da un file ini delle informazioni di configurazione nel costruttore di una scheda e di scrittura dei valori nel gestore di evento *OnClose*.

```
__fastcall TForm1::TForm1(TComponent *Owner) : TForm(Owner)
{
   TIniFile *ini;
   ini = new TIniFile( ChangeFileExt( Application->ExeName, ".INI" ) );

   Top     =  ini->ReadInteger( "Form", "Top", 100 );
   Left    =  ini->ReadInteger( "Form", "Left", 100 );
   Caption =  ini->ReadString( "Form", "Caption",
                              "Default Caption" );
   ini->ReadBool( "Form", "InitMax", false ) ?
         WindowState = wsMaximized :
         WindowState = wsNormal;

   delete ini;
}

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
   TIniFile *ini;
   ini = new TIniFile(ChangeFileExt( Application->ExeName, ".INI" ) );
   ini->WriteInteger( "Form", "Top", Top );
   ini->WriteInteger( "Form", "Left", Left );
   ini->WriteString ( "Form", "Caption", Caption );
   ini->WriteBool   ( "Form", "InitMax",
                      WindowState == wsMaximized );
```

```
        delete ini;
    }
```

Ognuna delle routine Read accetta tre parametri. Il primo parametro identifica la sezione del file ini. Il secondo parametro identifica il valore che si desidera leggere e il terzo è un valore predefinito nel caso la sezione o il valore non esista nel file ini. Esattamente come i metodi Read gestiscono le situazioni in cui una sezione o un valore non esistono, le routine Write creano la sezione e/o il valore nel caso esse non esistano. La prima volta che viene eseguito, il codice di esempio crea un file INI simile al seguente:

```
[Form]
Top=185
Left=280
Caption=Default Caption
InitMax=0
```

Alle successive esecuzioni dell'applicazione, i valori del file ini saranno letti al momento della creazione della scheda, e riscritti nel file nell'evento *OnClose*.

Utilizzo di TRegistryIniFile

La maggior parte delle applicazioni a 32-bit memorizzano le proprie informazioni nel file di registro invece che nel file ini poiché il file di registro è gerarchico e non è soggetto ai limiti sulle dimensione dei file ini. Se si è abituati ad utilizzare i file ini e si vogliono spostare invece le informazioni di configurazione nel file di registro, è possibile utilizzare la classe *TRegistryIniFile*. Si potrebbe anche volere utilizzare *TRegistryIniFile* in applicazioni multipiattaforma nel caso si voglia utilizzare il Registro di sistema in Windows e un file ini in Linux. È possibile scrivere la maggior parte dell'applicazione in modo che utilizzi il tipo *TCustomIniFile*. È necessario solo condizionare il codice che crea un'istanza di *TRegistryIniFile* (in Windows) o *TMemIniFile* (in Linux) e assegnarla al *TCustomIniFile* utilizzato dall'applicazione.

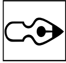
TRegistryIniFile è progettata per far sì che le voci del file di registro assomiglino alle voci del file INI. Tutti i metodi di *TIniFile* e di *TMemIniFile* (lettura e scrittura) esistono in *TRegistryIniFile*.

Quando si costruisce un oggetto *TRegistryIniFile* , il parametro che si passa (il nome file per un oggetto *IniFile* oppure un oggetto TmemIniFile) diventa un valore di chiave nelle chiavi utente nel file di registro e tutte le sezioni e i valori dipendono da quella radice. Tutte le sezioni e i valori sono ramificazioni di quella radice. *TRegistryIniFile* semplifica notevolmente l'interfaccia del file di registro, tanto che lo si utilizzerà al posto del componente di *TRegistry*, anche se non si sta convertendo del codice esistente o scrivendo un'applicazione cross-platform.

Uso di TRegistry

Se si sta scrivendo un'applicazione solamente per Windows e la struttura del Registro di sistema è familiare, è possibile utilizzare *TRegistry*. A differenza di *TRegistryIniFile*, che utilizza le stesse proprietà e metodi degli altri componenti per file ini, le proprietà e i metodi di *TRegistry* corrispondono più direttamente alla struttura del Registro di sistema. Ad esempio, *TRegistry* permette di specificare sia la

chiave radice che la sottochiave, mentre *TRegistry* assume HKEY_CURRENT_USER come chiave radice. Oltre ai metodi per aprire, chiudere, salvare, spostare, copiare e cancellare chiavi, *TRegistry* permette di specificare il livello di accesso che si desidera utilizzare.



TRegistry non è utilizzabile per programmi multipiattaforma.

L'esempio seguente recupera un valore da una voce del file di registro:

```
#include <Registry.hpp>

AnsiString GetRegistryValue(AnsiString KeyName)
{
  AnsiString S;
  TRegistry *Registry = new TRegistry(KEY_READ);
  try
  {
    Registry->RootKey = HKEY_LOCAL_MACHINE;
    // False because we do not want to create it if it doesn't exist
    Registry->OpenKey(KeyName,false);
    S = Registry->ReadString("VALUE1");
  }
  __finally
  {
    delete Registry;
  }
  return S;
}
```

Operazioni con le liste

BaseCLX include molte classi che rappresentano elenchi o raccolte di elementi. Esse variano a seconda dei tipi di elementi che contengono, delle operazioni che supportano e dal fatto che siano o meno persistenti.

La seguente tabella elenca diverse classi di elenco e indica i tipi di elementi che contengono:

Tabella 4.5 Classi per la gestione di elenchi

Oggetto	Significato
TList	Elenco di puntatori
TThreadList	Un elenco thread-safe di puntatori
TBucketList	Un elenco hashed di puntatori
TObjectBucketList	Un elenco hashed di istanze di oggetti
TObjectList	Elenco di istanze di oggetti gestito in memoria
TComponentList	Elenco di componenti gestito in memoria (ovvero, istanze di classi derivate da *TComponent*)
TClassList	Un elenco di riferimenti di classe (metaclassi)
TInterfaceList	Un elenco di puntatori a interfacce.
TQueue	Elenco di puntatori FIFO (first-in, first-out)

Tabella 4.5 Classi per la gestione di elenchi

Oggetto	Significato
TStack	Elenco di puntatori LIFO (last-in, first-out)
TObjectQueue	Elenco di oggetti FIFO (first-in, first-out)
TObjectStack	Elenco di oggetti LIFO (last-in, first-out)
TCollection	Classe base per molte classi specializzate di elementi con tipo.
TStringList	Elenco di stringhe
THashedStringList	Un elenco di stringhe con nel formato Name=Value, hashed per prestazioni.

Operazioni comuni su elenchi

Benché le diverse classi di elenco contengano vari tipi di elementi e abbiano vari progenitori, la maggior parte di loro condivide un insieme comune di metodi per aggiungere, cancellare, riordinare e accedere agli elementi nell'elenco.

Aggiunta di elementi di un elenco

La maggior parte delle classi di elenco hanno un metodo *Add*, che permette di aggiungere un elemento alla fine dell'elenco (se non è ordinato) o nella posizione appropriata (se l'elenco è ordinato). Di solito, il metodo *Add* accetta come parametro l'elemento che si aggiunge all'elenco e restituisce la posizione nell'elenco in cui è stato aggiunto l'elemento. Nel caso di bucket lists (*TBucketList* e *TObjectBucketList*), *Add* accetta non solo l'elemento da aggiungere, ma anche un dato che è possibile associare a quell'elemento. Nel caso di collection, *Add* non accetta alcun parametro, ma crea un nuovo elemento che aggiunge. Il metodo *Add* sulle collection restituisce l'elemento che ha aggiunto, in modo che sia possibile assegnare valori alle proprietà del nuovo elemento.

Oltre al metodo *Add*, alcune classi di elenco hanno un metodo *Insert*. *Insert* funziona esattamente come il il metodo *Add*, ma ha un parametro aggiuntivo che consente di specificare la posizione nell'elenco in cui si desidera che venga visualizzato il nuovo elemento. Se una classe ha un metodo *Add*, ha anche un metodo *Insert* a meno che la posizione degli elementi non sia predeterminata. Ad esempio, è impossibile utilizzare *Insert* con elenchi ordinati poiché gli elementi devono essere messi secondo la sequenza di ordinamento ed è impossibile utilizzare *Insert* con bucket lists poiché l'algoritmo hash determina la posizione dell'elemento.

Le uniche classi che non hanno un metodo *Add* sono gli elenchi ordinati. Gli elenchi ordinati sono code e stack. Per aggiungere elementi a un elenco ordinato, utilizzare invece il metodo *Push*. *Push*, come *Add*, accetta un elemento come parametro e lo inserisce nella posizione corretta.

Cancellazione degli elementi di un elenco

Per cancellare un singolo elemento da una delle classi di elenco, utilizzare il metodo *Delete* oppure il metodo *Remove*. *Delete* accetta un solo parametro, il nome dell'elemento da rimuovere. Anche *Remove* accetta un singolo parametro ma quel parametro è un riferimento all'elemento da rimuovere, invece del suo indice. Alcune

classi di elenco supportano solo un metodo *Delete*, alcune supportano solo un metodo *Remove* e alcune li hanno entrambi.

Come con l'aggiunta di elementi, gli elenchi ordinati si comportano diversamente di tutti gli altri elenchi. Invece di utilizzare un metodo *Delete* o *Remove*, si rimuove un elemento da un elenco ordinato chiamando il metodo *Pop*. *Pop* non accetta alcun argomento, in quanto c'è solo un elemento che può essere rimosso.

Se si vogliono cancellare tutti gli elementi nell'elenco, è possibile chiamare il metodo *Clear*. La cancellazione è disponibile per tutti gli elenchi tranne quelli ordinati.

Accesso agli elementi di un elenco

Tutte le classi di elenco (tranne *TThreadList* e gli elenchi ordinati) hanno una proprietà che permette di accedere agli elementi nell'elenco. Di solito, questa proprietà è chiamata *Items*. Per gli elenchi di stringhe, la proprietà è chiamata *Strings* e per elenchi bucket è chiamata *Data*. Le proprietà *Items*, *Strings* o *Data* sono proprietà indicizzate e pertanto è possibile specificare a quale elemento si desidera accedere.

Su *TThreadList*, è necessario bloccare l'elenco prima di potere accedere agli elementi. Quando si blocca l'elenco, il metodo *LockList* restituisce un oggetto *TList* che è possibile utilizzare per accedere agli elementi.

Solo gli elenchi ordinati permettono di accedere all'elemento "superiore" dell'elenco. È possibile ottenere un riferimento a questo elemento chiamando il metodo *Peek*.

Riordinamento degli elementi di un elenco

Alcune classi di elenco hanno metodi che permettono di riordinare gli elementi nell'elenco. Alcune hanno un metodo *Exchange*, che scambia la posizione di due elementi. Alcune hanno un metodo *Move* che permette di spostare un elemento in una posizione specificata. Alcune hanno un metodo *Sort* che permette di ordinare gli elementi nell'elenco.

Per vedere quali metodi sono disponibili, controllare la Guida in linea relativamente alla classe di elenco che si sta utilizzando.

Elenchi persistenti

Gli elenchi persistenti possono essere salvati in un file scheda. Per questo motivo, sono speeso utilizzati come tipi di una proprietà pubblicata di un componente. È possibile aggiungere elementi all'elenco in fase di progettazione e quegli elementi vengono salvati con l'oggetto in modo che essi siano presenti quando il componente che li utilizza viene caricato in memoria in fase esecuzione. Ci sono due tipi principali di elenchi persistenti: elenchi di stringhe e collection.

Come esempi di elenchi di stringhe si possono citare *TStringList* e *THashedStringList*. Gli elenchi di stringhe, come dice anche il nome, contengono stringhe. Essi forniscono un supporto speciale per stringhe in formato Name=Value, in modo che sia possibile reperire il valore associato a un nome. Inoltre, la maggior parte degli elenchi di stringhe permettono di associare un oggetto a ogni stringa nell'elenco. Le

liste di stringhe sono descritte in modo più dettagliato in "Operazioni con elenchi di stringhe" a pagina 4-16.

Le collection discendono dalla classe *TCollection*. Ogni discendente *TCollection* è specializzato nella gestione di una specifica classe di elementi, dove quella classe discende da *TCollectionItem*. Le collection supportano la maggior parte delle comuni operazioni sugli elenchi. Tutte le collection sono progettate per essere del tipo di una proprietà pubblicata, e molte non possono funzionare in modo indipendente dall'oggetto che le utilizza per implementare una delle sue proprietà. In progettazione, la proprietà il cui valore è una raccolta può utilizzare il Collection editor l'editor per consentire l'aggiunt , la rimozione e il riordino di elementi. Il Collection editor fornisce un'interfaccia utente comune per il trattamento delle collection

Operazioni con elenchi di stringhe

Uno dei tipi di elenco più comunemente utilizzato è un elenco di stringhe di caratteri. Alcuni esempi potrebbero essere gli elementi di una casella combinata, le righe di un memo, i nomi dei tipi di carattere o i nomi di righe e colonne in una griglia per stringhe. La VCL mette a disposizione un'interfaccia comune per un qualsiasi elenco di stringhe mediante un oggetto di nome *TStrings* e mediante derivati come *TStringList* e THashedStringlist. *TStringList* implementa le proprietà e i metodi astratti introdotti da *TStrings* e introduce proprietà, eventi e metodi per

- Ordinare le stringhe nell'elenco.
- Evitare stringhe duplicate negli elenchi ordinati.
- Rispondere alle modifiche apportate ai contenuti dell'elenco.

Oltre alle funzionalità per gestire gli elenchi di stringhe, questi oggetti consentono un'interazione oltremodo semplice; ad esempio, è possibile modificare le righe di un memo, (che è un'istanza di *TStrings*) e quindi utilizzare queste righe come elementi di una casella combinata (che è anch'essa un'istanza di *TStrings*).

Con gli oggetti *TStrings* nella colonna Value dell'Object Inspector è disponibile una proprietà per gestire gli elenchi di stringhe. Fare doppio clic su *TStrings* per aprire lo String List editor, in cui è possibile modificare, aggiungere o cancellare righe.

Con gli oggetti string-list è possibile operare anche in fase di esecuzione per eseguire operazioni quali

- Caricamento e salvataggio di elenchi di stringhe
- Creazione di un nuovo elenco di stringhe
- Gestione di stringhe in un elenco
- Associazione di oggetti ad un elenco di stringhe

Caricamento e salvataggio di elenchi di stringhe

Gli oggetti string-list mettono a disposizione metodi *SaveToFile* e *LoadFromFile* che consentono di memorizzare un elenco di stringhe in un file di testo e caricano un file di testo in un elenco di stringhe. Ciascuna riga del file di testo corrisponde a una

stringa dell’elenco. Utilizzando questi metodi, è possibile, per esempio, creare un semplice editor di testi caricando un file in un componente memo, oppure salvare liste di elementi per caselle combinate.

L’esempio seguente carica una copia del file WIN.INI in un campo memo ed esegue una copia di riserva di nome WIN.BAK.

```
void __fastcall EditWinIni()
{
  AnsiString FileName = "C:\\WINDOWS\\WIN.INI";// set the file name
  Form1->Memo1->Lines->LoadFromFile(FileName);      // load from file
  Form1->Memo1->Lines->SaveToFile(ChangeFileExt(FileName, ".BAK"));  // save to backup
}
```

Creazione di un nuovo elenco di stringhe

Un elenco di stringhe di solito fa parte di un componente. A volte, tuttavia, risulta comodo creare elenchi di stringhe indipendenti, ad esempio per memorizzare stringhe per una tabella di consultazione. Il modo in cui si crea e si gestisce un elenco di stringhe dipende dal fatto che l’elenco sia a breve termine (creato, usato e distrutto in una singola routine) oppure a lungo termine (disponibile finché l’applicazione non viene terminata). Indipendentemente dal tipo di elenco di stringhe creato, al termine delle operazioni occorre ricordarsi di liberare la memoria utilizzata.

Elenchi di stringhe a breve termine

Se si utilizza un elenco di stringhe solo per la durata di una singola routine, è possibile crearlo, utilizzarlo e distruggerlo in un solo posto. Questo è il modo più sicuro per lavorare con elenchi di stringhe. Poiché l’oggetto string list alloca memoria per sé e per le proprie stringhe, per essere certi che la memoria venga liberata anche nel caso si verifichi un’eccezione si dovrebbe utilizzare un blocco **try...__finally**.

1 Costruire l’oggetto string list.
2 Nella parte **try** di un blocco **try...__finally**, utilizzare l’elenco di stringhe.
3 Nella parte **__finally**, liberare l’oggetto string list.

Il seguente gestore di evento risponde al clic di un pulsante costruendo un elenco di stringhe, utilizzandolo, e quindi distruggendolo.

```
void __fastcall TForm1::ButtonClick1(TObject *Sender)
{
   TStringList *TempList = new TStringList; // declare the list
   try{
      //use the string list
   }
   __finally{
    delete TempList; // destroy the list object
   }
}
```

Elenchi di stringhe a lungo termine

Se un elenco di stringhe deve essere disponibile in un qualunque momento durante l'esecuzione dell'applicazione, costruire l'elenco all'avvio dell'applicazione e distruggerlo prima che l'applicazione termini.

1. Nel file unit della scheda principale dell'applicazione, aggiungere un campo di tipo *TStrings* alla dichiarazione della scheda.
2. Scrivere un *costruttore* per la scheda principale che viene eseguito prima che la scheda appaia. Dovrebbe creare un elenco di stringhe e dovrebbe assegnarlo al campo dichiarato al punto precedente.
3. Scrivere un gestore di evento che liberi l'elenco di stringhe nell'evento *OnClose* della scheda.

Questo esempio utilizza un elenco di stringhe a lungo termine per registrare i clic che l'utente fa con il mouse sulla scheda principale, quindi salva l'elenco in un file prima del termine dell'applicazione.

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
       : TForm(Owner)
{
   ClickList = new TStringList;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
   ClickList->SaveToFile(ChangeFileExt(Application->ExeName, ".LOG"));//Save the list
   delete ClickList;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
   TVarRec v[] = {X,Y};
   ClickList->Add(Format("Click at (%d, %d)",v,ARRAYSIZE(v) - 1));//add a string to the list
}
}
```

Gestione di stringhe in un elenco

Le operazioni compiute di solito sugli elenchi di stringhe includono:

- Conteggio di stringhe in un elenco

- Accesso ad una particolare stringa
- Ricerca della posizione di una stringa nell'elenco
- Analisi sequenziale di stringhe in un elenco
- Aggiunta di una stringa a un elenco
- Spostamento di una stringa all'interno di un elenco
- Cancellazione di una stringa da un elenco
- Copia completa di un elenco di stringhe

Conteggio di stringhe in un elenco

La proprietà a sola lettura *Count* restituisce il numero di stringhe nell'elenco. Poiché gli elenchi di stringhe utilizzano indici con base zero, il valore di *Count* è pari all'indice dell'ultima stringa più uno.

Accesso ad una particolare stringa

La proprietà array *Strings* contiene le stringhe dell'elenco, referenziate da un indice con base zero.

```
StringList1->Strings[0] = "This is the first string.";
```

Individuazione di elementi in un elenco di stringhe

Per individuare una stringa in un elenco di stringhe, utilizzare il metodo *IndexOf*. *IndexOf* restituisce l'indice della prima stringa nell'elenco che è uguale al parametro passatogli, e restituisce –1 se la stringa parametro non viene trovata. *IndexOf* trova solo le corrispondenze esatte; se si desidera cercare una corrispondenza di stringa parziale, si deve scrivere del codice che esamini in sequenza l'elenco di stringhe.

Per esempio, si potrebbe utilizzare *IndexOf* per determinare se il nome di un certo file è presente negli Elements di una casella di riepilogo:

```
if (FileListBox1->Items->IndexOf("WIN.INI") > -1) ...
```

Analisi sequenziale di stringhe in un elenco

Per esaminare in modo sequenziale le stringhe in un elenco, utilizzare un ciclo **for** che parta da zero e arrivi a *Count*–1.

Questo esempio converte ciascuna stringa in una casella di riepilogo in caratteri maiuscoli.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  for (int i = 0; i < ListBox1->Items->Count; i++)
    ListBox1->Items->Strings[i] = UpperCase(ListBox1->Items->Strings[i]);
}
```

Aggiunta di una stringa a un elenco

Per aggiungere una stringa alla fine di un elenco di stringhe, chiamare il metodo *Add*, passando la nuova stringa come parametro. Per inserire una stringa nell'elenco, chiamare il metodo *Insert* passando due parametri: la stringa e l'indice della posizione in cui la si vuole collocare. Per esempio, per fare in modo che la stringa

"Three" sia inserita nella terza posizione in un elenco si potrebbe utilizzare l'istruzione:

```
StringList1->Insert(2, "Three");
```

Per aggiungere le stringhe di un elenco ad un altro, chiamare il metodo *AddStrings*:

```
StringList1->AddStrings(StringList2);  // append the strings from StringList2 to StringList1
```

Spostamento di una stringa all'interno di un elenco

Per spostare una stringa in un elenco di stringhe, chiamare il metodo *Move* passando due parametri: l'indice corrente della stringa e l'indice che si vuole assegnare. Per esempio, per spostare la terza stringa in un elenco alla quinta posizione si potrebbe utilizzare:

```
StringListObject->Move(2, 4);
```

Cancellazione di una stringa da un elenco

Per cancellare una stringa da un elenco di stringhe, chiamare il metodo *Delete* dell'elenco, passando l'indice della stringa che si desidera cancellare. Se non si conosce qual è l'indice della stringa che si vuole cancellare, utilizzare il metodo *IndexOf* per individuarlo. Per cancellare tutte le stringhe in un elenco di stringhe, utilizzare il metodo *Clear*.

Questo esempio utilizza *IndexOf* e *Delete* per trovare e cancellare una stringa:

```
int BIndex = ListBox1->Items->IndexOf("bureaucracy");
if (BIndex > -1)
  ListBox1->Items->Delete(BIndex);
```

Copia di un intero elenco di stringhe

Si può utilizzare il metodo *Assign* per copiare le stringhe da un elenco sorgente in un elenco di destinazione, sovrascrivendo i contenuti dell'elenco di destinazione. Per aggiungere le stringhe senza sovrascrivere l'elenco di destinazione, utilizzare *AddStrings*. Ad esempio:

```
Memo1->Lines->Assign(ComboBox1->Item)s; //overwrites original strings
```

copia le righe da una casella combinata in un memo (sovrascrivendo il memo), mentre

```
Memo1->Lines->AddStrings(ComboBox1->Items);//appends strings to end
```

aggiunge al memo le righe contenute nella casella combinata.

Quando si fanno copie locali di un elenco di stringhe, utilizzare il metodo *Assign*. Quando si aggiunge una variabile string list ad un'altra—

```
StringList1 = StringList2;
```

—l'oggetto string list originale andrà perduto, spesso con risultati imprevisti.

Associazione di oggetti ad un elenco di stringhe

Oltre alle stringhe memorizzate nella proprietà *Strings*, un elenco di stringhe può conservare riferimenti a oggetti che vengono memorizzati nella proprietà *Objects*.

Come *Strings*, anche *Objects* è un array con un indice a base zero. L'uso più comune di *Objects* è quello di associare bitmap a stringhe per controlli con tracciamento personalizzato.

Utilizzare i metodi *AddObject* o *InsertObject* per aggiungere all'elenco una stringa e un oggetto associato in una sola passata. *IndexOfObject* restituisce l'indice della prima stringa nell'elenco che è associata ad un oggetto specificato. Metodi come *Delete*, *Clear* e *Move* operano contemporaneamente su stringhe e oggetti; ad esempio, la cancellazione di una stringa rimuove l'oggetto corrispondente (se ne esiste uno).

Per associare un oggetto con una stringa esistente, assegnare l'oggetto alla proprietà *Objects* usando lo stesso indice. Non è possibile aggiungere un oggetto senza aggiungere prima una stringa corrispondente.

Operazioni con stringhe

La libreria runtime BaseCLX fornisce molte routine specialistiche di gestione delle stringhe, specifiche per un tipo stringa. Queste sono routine per stringhe wide, Ansi e null-terminated (char*). Le routine che hanno a che fare con i tipi null-terminated usano la terminazione con null per determinare la lunghezza della stringa. I seguenti argomenti forniscono una panoramica su molte routine per la gestione delle string incluse nella libreria di esecuzione.

Routine per wide character

Le wide string sono utilizzate in un moltissime situazioni. Alcune tecnologie, come XML, utilizzano le wide string come tipo nativo. È anche possibile scegliere di utilizzare le wide string in quanto semplificano alcune problematiche connesse alla gestione delle stringhe in applicazioni che hanno più ambienti di destinazione. L'uso di uno schema di codifica con caratteri estesi offre il vantaggio di poter effettuare molte delle solite assunzioni sulle stringhe che non funzionano per i sistemi MBCS. C'è una relazione diretta tra il numero di byte della stringa e il suo numero di caratteri. Non ci si deve preoccupare di troncare i caratteri a metà o di confondere la seconda metà di un carattere con l'inizio di un altro.

Uno svantaggio nell'utilizzo dei caratteri estesi è che la maggior parte dei controlli della VCL rappresentano valori di stringa come stringhe a singolo byte o stringhe MBCS . (I controlli CLX di solito utilizzano wide string.) La conversione dal sistema wide character al sistema MBCS, e viceversa, ogni volta che si imposta una proprietà stringa o se ne legge il valore, richiederebbe notevoli quantità di ulteriore codice e rallenterebbe l'applicazione. Tuttavia, può risultare necessario tradurre in wide character alcuni speciali algoritmi di elaborazione delle stringhe che hanno bisogno di sfruttare i vantaggi della correlazione 1:1 tra caratteri e *WideChars*.

Le seguenti funzioni permettono la conversione tra le stringhe di caratteri standard a singolo byte (o stringhe MBCS) e le stringhe Unicode:

- StringToWideChar
- WideCharLenToString

- WideCharLenToStrVar
- WideCharToString
- WideCharToStrVar

Inoltre, le seguenti funzioni effettuano la traduzione da WideStrings a altre rappresentazioni e viceversa:

- UCS4StringToWideString
- WideStringToUCS4String
- VarToWideStr
- VarToWideStrDef

Le seguenti routine operano direttamente con WideStrings:

- WideCompareStr
- WideCompareText
- WideSameStr
- WideSameText
- WideSameCaption (CLX only)
- WideFmtStr
- WideFormat
- WideLowerCase
- WideUpperCase

Infine, alcune routine includono degli overload per operare con le wide string:

- UniqueString
- Length
- Trim
- TrimLeft
- TrimRight

Routine comunemente utilizzate per AnsiStrings

Le routine di gestione delle AnsiString coprono diverse aree funzionali. All'interno di queste aree, alcune vengono usate per lo stesso scopo, differenziandosi solo per il fatto che usano o meno un particolare criterio nei calcoli. Le tabelle riportate di seguito elencano queste routine suddividendole in queste aree funzionali:

- Confronto
- Conversione tra lettere maiuscole/minuscole
- Modifica
- Sottostringa

Quando è possibile, le tabelle forniscono anche le colonne per indicare se una routine soddisfa i seguenti criteri.

- Sensibilità a maiuscole/minuscole: Se si utilizzano le impostazioni locali, determina la definizione del formato delle lettere. Se la routine non usa le impostazioni locali, l'analisi viene effettuata sui valori ordinali dei caratteri. Se la routine tiene conto della differenza tra lettere maiuscole/ minuscole, c'è una fusione logica dei caratteri maiuscoli e minuscoli che viene determinata da un modello predefinito.

- Usa impostazioni locali: L'abilitazione della versione locale consente di personalizzare l' applicazione per specifiche nazioni, in particolare per gli ambienti con lingue dell'Asia. La maggior parte delle versioni nazionalizzate considera i caratteri minuscoli minori dei corrispondenti caratteri maiuscoli. Questo è in contrasto con l'ordinamento ASCII, nel quale i caratteri minuscoli sono maggiori dei caratteri maiuscoli. Le routine che usano le impostazioni locali di Windows sono normalmente precedute dalla parola Ansi (cioè, *AnsiXXX*).
- Supporta il set di caratteri multibyte (MBCS): I MBCS vengono usati quando si scrive codice per paesi dell'estremo oriente. I caratteri multi-byte vengono rappresentati come un combinazione di codici di carattere ad uno e a due byte, cosicché la lunghezza in byte non corrisponde necessariamente alla lunghezza della stringa. Le routine che supportano i MBCS sono scritte per analizzare caratteri ad uno e a due byte.

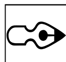
  ByteType e *StrByteType* determinano se un particolare byte è quello iniziale di un carattere di due byte. Quando si usano i caratteri multi-byte bisogna prestare attenzione a non troncare una stringa tagliando a metà un carattere di due byte. Non si devono passare caratteri come parametro ad una funzione o procedura, poiché le dimensioni di un carattere non possono essere predeterminate. Si può passare, invece, un puntatore ad un carattere o ad una stringa. Per ulteriori informazioni su MBCS, consultare il paragrafo "Abilitazione del codice dell'applicazione" a pagina 16-2 of Capitolo 16, "Creazione di applicazioni internazionali".

Tabella 4.6 Routine per il confronto delle stringhe

Routine	Sensibilità a maiuscole/minuscole	Usa impostazioni locali	Supporta MBCS
AnsiCompareStr	Sì	Sì	Sì
AnsiCompareText	no	Sì	Sì
AnsiCompareFileName	no	Sì	Sì
AnsiMatchStr	Sì	Sì	yes
AnsiMatchText	no	Sì	Sì
AnsiContainsStr	Sì	Sì	Sì
AnsiContainsText	no	Sì	Sì
AnsiStartsStr	Sì	Sì	Sì
AnsiStartsText	no	Sì	Sì
AnsiEndsStr	Sì	Sì	Sì
AnsiEndsText	no	Sì	Sì
AnsiIndexStr	Sì	Sì	Sì
AnsiIndexText	no	Sì	Sì
CompareStr	Sì	no	no
CompareText	no	no	no
AnsiResemblesText	no	no	no

Tabella 4.7 Routine di conversione maiuscole/minuscole

Routine	Usa impostazioni locali	Supporta MBCS
AnsiLowerCase	Sì	Sì
AnsiLowerCaseFileName	Sì	Sì
AnsiUpperCaseFileName	Sì	Sì
AnsiUpperCase	Sì	Sì
LowerCase	no	no
UpperCase	no	no



Le routine usate per le stringhe dei nomi di file: *AnsiCompareFileName*, AnsiLowerCaseFileNamee *AnsiUpperCaseFileName* usano tutti le impostazioni locali di Windows. Occorre usare sempre nomi di file che siano perfettamente trasportabili, dal momento che la lingua (il set di caratteri) usato per questi nomi può e potrebbe differire dall'interfaccia utente predefinita.

Tabella 4.8 Routine per la modifica di stringhe

Routine	Sensibilità a maiuscole/ minuscole	Supporta MBCS
AdjustLineBreaks	ND	Sì
AnsiQuotedStr	ND	Sì
AnsiReplaceStr	Sì	Sì
AnsiReplaceText	no	Sì
StringReplace	optional by flag	Sì
ReverseString	ND	no
StuffString	ND	no
Trim	ND	Sì
TrimLeft	ND	Sì
TrimRight	ND	Sì
WrapText	ND	Sì

Tabella 4.9 Routine per le sottostringhe

Routine	Sensibilità a maiuscole/ minuscole	Supporta MBCS
AnsiExtractQuotedStr	ND	Sì
AnsiPos	Sì	Sì
IsDelimiter	Sì	Sì
IsPathDelimiter	Sì	Sì
LastDelimiter	Sì	Sì

Tabella 4.9 Routine per le sottostringhe

Routine	Sensibilità a maiuscole/ minuscole	Supporta MBCS
LeftStr	ND	no
RightStr	ND	no
MidStr	ND	no
QuotedStr	no	no

Routine comunemente utilizzate per stringhe terminate con null

Le routine di gestione delle stringhe terminate con null coprono diverse aree funzionali. All'interno di queste aree, alcune vengono usate per lo stesso scopo, differenziandosi solo per il fatto che usano o meno un particolare criterio nei calcoli. Le tabelle riportate di seguito elencano queste routine suddividendole in queste aree funzionali:

- Confronto
- Conversione tra lettere maiuscole/minuscole
- Modifica
- Sottostringa
- Copia

Ove opportuno, le tabelle forniscono anche colonne che indicano se la routine è sensibile alle maiuscole/minuscole, se utilizza l'ambiente corrente e/o supporta i set di caratteri multibyte.

Tabella 4.10 Routine di confronto per stringhe terminate con null

Routine	Sensibilità a maiuscole/minuscole	Usa impostazioni locali	Supporta MBCS
AnsiStrComp	Sì	Sì	Sì
AnsiStrIComp	no	Sì	Sì
AnsiStrLComp	Sì	Sì	Sì
AnsiStrLIComp	no	Sì	Sì
StrComp	Sì	no	no
StrIComp	no	no	no
StrLComp	Sì	no	no
StrLIComp	no	no	no

Tabella 4.11 Routine di conversione maiuscole/minuscole per stringhe terminate con null

Routine	Usa impostazioni locali	Supporta MBCS
AnsiStrLower	Sì	Sì
AnsiStrUpper	Sì	Sì

Tabella 4.11 Routine di conversione maiuscole/minuscole per stringhe terminate con null

Routine	Usa impostazioni locali	Supporta MBCS
StrLower	no	no
StrUpper	no	no

Tabella 4.12 Routine per la modifica di stringhe

Routine
StrCat
StrLCat

Tabella 4.13 Routine per le sottostringhe

Routine	Sensibilità a maiuscole/ minuscole	Supporta MBCS
AnsiStrPos	Sì	Sì
AnsiStrScan	Sì	Sì
AnsiStrRScan	Sì	Sì
StrPos	Sì	no
StrScan	Sì	no
StrRScan	Sì	no

Tabella 4.14 Routine per la copia di stringhe

Routine
StrCopy
StrLCopy
StrECopy
StrMove
StrPCopy
StrPLCopy

Stampa

Dal punto di vista strettamente tecnico, la classe *TPrinter* non appartiene a BaseCLX in quanto vi sono due versioni separate, una per la VCL (nella unit *Printers*) e una per la CLX (nella unit QPrinters). L'oggetto *TPrinter* della VCL incapsula i dettagli delle stampanti Windows. L'oggetto *TPrinter* è un dispositivo per il disegno su stampante. Esso genera del codice postscript e lo invia ai dispositivi lpr, lp o a un altro comando di stampa. Entrambe le versioni di *TPrinter*, tuttavia, sono estremamente simili.

Per ottenere un elenco delle stampanti installate disponibili, utilizzare la proprietà *Printers*. Entrambi gli oggetti printer utilizzano *TCanvas* (identico al *TCanvas* della scheda), il che significa che tutto ciò che può essere tracciato su una scheda può anche

essere stampato. Per stampare un'immagine, chiamare il metodo *BeginDoc* seguito da un qualsiasi canvas grafico che si desidera stampare (incluso del testo mediante il metodo *TextOut*) e inviare il lavoro alla stampante chiamando il metodo *EndDoc*.

L'esempio seguente utilizza un pulsante e un campo memo su una scheda. Quando l'utente fa clic sul pulsante, il contenuto del memo viene stampato a una distanza di 200-pixel dal bordo della pagina.

Per eseguire con successo questo esempio, includere <Printers.hpp> nel file unit.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TPrinter *Prntr = Printer();
  TRect r = Rect(200,200,Prntr->PageWidth - 200,Prntr->PageHeight- 200);
  Prntr->BeginDoc();
  for( int i = 0; i < Memo1->Lines->Count; i++)
    Prntr->Canvas->TextOut(200,200 + (i *
  Prntr->Canvas->TextHeight(Memo1->Lines->Strings[i])),
  Memo1->Lines->Strings[i]);
  Prntr->Canvas->Brush->Color = clBlack;
  Prntr->Canvas->FrameRect(r);
  Prntr->EndDoc();
}
```

Per ulteriori informazioni sull'utilizzo dell'oggetto *TPrinter*, consultare la Guida in linea alla voce *TPrinter*.

Conversione di misure

La unit ConvUtils dichiara una funzione *Convert* di uso generale che è possibile utilizzare per convertire una misura da un sistema di misura a uno altro. È possibile eseguire conversioni tra unità di misura compatibili come piedi e pollici o giorni e settimane. Le unità di misura che misurano lo stesso tipo di cose sono dette appartenere alla stessa *famiglia di conversione*. Le unita che devono essere convertite devono appartenere alla stessa famiglia di conversione. Per informazioni su come effettuare una conversione, vedere la sezione successiva Esecuzione di conversioni e consultare l'argomento *Convert* nella Guida in linea.

La unit StdConvs definisce diverse famiglie di conversione e unità di misura all'interno di ogni famiglia. Inoltre, è possibile creare famiglie di conversione personalizzate e le relative unità di misura utilizzando le funzioni *RegisterConversionType* e *RegisterConversionFamily*.Per informazioni sull'ampliamento delle conversioni e sulle unita di conversione, consultare la sezione Aggiunta di nuovi tipi di misura e consultare l'argomento *Convert* nella Guida in linea.

Esecuzione di conversioni

È possibile utilizzare la funzione *Convert* per eseguire conversioni sia semplici che complesse. La funzione ha una sintassi semplificata e una seconda sintassi per l'esecuzione di conversioni tra tipi di misura complessi.

Esecuzione di conversioni semplici

È possibile utilizzare la funzione *Convert* per convertire una misura da un insieme di unità a un altro. La funzione *Convert* serve per convertire unità di misura che misurano lo stesso tipo di cose (distanze, superfici, tempo, temperature e così via).

Per utilizzare *Convert*, è necessario specificare le unità di misura da cui convertire e in cui convertire. Si utilizza il tipo *TConvType* per identificare le unita di misura.

Ad esempio, queste istruzioni convertono una temperatura da gradi Fahrenheit a gradi Kelvin:

```
TempInKelvin = Convert(StrToFloat(Edit1->Text), tuFahrenheit, tuKelvin);
```

Esecuzione di conversioni complesse

È possibile utilizzare la funzione *Convert* anche per eseguire conversioni più complesse tra il rapporto di due tipi di misura. Un esempio della necessità di ricorrere a questo tipo di conversione potrebbe essere la conversione da miglia orarie a metri al minuto per il calcolo della velocità oppure la conversione da galloni al minuto a litri per ora per il calcolo del flusso.

Ad esempio, la seguente chiamata converte da miglia per gallone in chilometri per litro:

```
double nKPL = Convert(StrToFloat(Edit1.Text), duMiles, vuGallons, duKilometers, vuLiter);
```

Le unità di misura che devono essere convertite devono appartenere alla stessa famiglia di conversione (devono misurare la stessa cosa). Se le unità di misura non sono compatibili, *Convert* solleva un'eccezione *EConversionError*. È possibile controllare se due valori *TConvType* appartengono alla stessa famiglia di conversione chiamando *CompatibleConversionTypes*.

La unit StdConvs definisce diverse famiglie di valori *TConvType*. Per una lista delle famiglie predefinite di unità di misura e delle unità di misura in ogni famiglia, consultare l'argomento "Conversion family variables" nella Guida in linea.

Aggiunta di nuovi tipi di misura

Se si vogliono eseguire conversioni tra unità di misura che non sono definite nella unit StdConvs, è necessario creare una nuova famiglia di conversione per rappresentare le unità di misura (valori *TConvType*). Se due valori *TConvType* sono registrati con la stessa famiglia di conversione, la funzione *Convert* può effettuare la conversione di misurazioni fatte utilizzando le unità di misura rappresentate da quei valori *TConvType*.

Per prima cosa è necessario ottenere i valori *TConvFamily* mediante la registrazione di una famiglia di conversione utilizzando la funzione *RegisterConversionFamily*. Dopo avere ottenuto un valore *TConvFamily* (registrando una nuova famiglia di conversione o utilizzando una delle variabili globali nella unit *StdConvs*), è possibile utilizzare la funzione *RegisterConversionType* per aggiungere le nuove unità di misura alla famiglia di conversione. Gli esempi seguenti mostrano come fare.

Creazione di una semplice famiglia di conversione e aggiunta delle unità

Un esempio della necessità di creare una nuova famiglia di conversione e aggiungere nuovi tipi di unità di misura potrebbe essere la conversione fra lunghi periodi temporali (ad esempio, da mesi a secoli) in cui potrebbe verificarsi una perdita di precisione.

Per approfondire il problema, la famiglia *cbTime* utilizza come unità di misura di base il giorno. L'unità di misura di base è quella che viene utilizzata per eseguire tutte le conversioni all'interno di quella famiglia. Quindi, tutte le conversioni devono essere fatte in termini di giorni. Quando si eseguono conversioni utilizzando come unità di misura i mesi o una unità maggiore (anni, decenni, secoli, millenni) potrebbero verificarsi delle inesattezze perché non esiste una conversione esatta tra giorni e mesi, giorni e anni, e così via. I mesi hanno varie lunghezze; gli anni hanno fattori di correzione per scarti di anni, di secondi, e così via.

Se si utilizzano solo unità di misura maggiori o uguali ai mesi, è possibile creare una famiglia di conversione più precisa che utilizza gli anni come unità di misura di base.Questo esempio crea una nuova famiglia di conversione di nome *cbLongTime*.

Dichiarazione delle variabili

Per prima cosa è necessario dichiarare le variabili per gli identificativi. Gli identificativi sono utilizzati nella nuova famiglia di conversione LongTime e nelle unità di misura che sono membri della famiglia:

```
tConvFamily cbLongTime;
TConvType ltMonths;
TConvType ltYears;
TConvType ltDecades;
TConvType ltCenturies;
TConvType ltMillennia;
```

Registrazione della famiglia di conversione

Poi, si deve registrare la famiglia di conversione:

```
cbLongTime = RegisterConversionFamily (“Long Times”);
```

Benché sia disponibile una procedura *UnregisterConversionFamily*, non è necessario annullare la registrazione di famiglie di conversione a meno che la unit che le definisce non venga rimossa in fase di esecuzione. Esse vengono automaticamente annullate quando si chiude l'applicazione.

Registrazione delle unità di misura

Poi, è necessario registrare le unità di misura all'interno della famiglia di conversione appena creata. Si utilizza la funzione *RegisterConversionType*, che registra le unità di misura all'interno di una determinata famiglia.È necessario definire l'unità di misura di base, che nell'esempio è l'anno, e le altre unità di misura vengono definite utilizzando un fattore che indica la loro relazione con l'unità di misura di base. Così, il fattore per *ltMonths* è 1/12 perché l'unità di misura di base per la famiglia

LongTime è l'anno. Includere anche una descrizione delle unità di misura verso cui si converte.

Il codice per registrare le unità di misura è riportato di seguito:

```
ltMonths = RegisterConversionType(cbLongTime,"Months",1/12);
ltYears = RegisterConversionType(cbLongTime,"Years",1);
ltDecades = RegisterConversionType(cbLongTime,"Decades",10);
ltCenturies = RegisterConversionType(cbLongTime,"Centuries",100);
ltMillennia = RegisterConversionType(cbLongTime,"Millennia",1000);
```

Utilizzo delle nuove unità di misura

A questo punto è possibile utilizzare le unità di misura appena registrate per eseguire una conversione. La funzione globale *Convert* può effettuare la conversione tra tutti i tipi di conversione registrati con la famiglia di conversione *cbLongTime*.

Pertanto, invece di utilizzare la seguente chiamata a *Convert*,

```
Convert(StrToFloat(Edit1->Text),tuMonths,tuMillennia);
```

è possibile utilizzare ora questa chiamata per una maggior precisione:

```
Convert(StrToFloat(Edit1->Text),ltMonths,ltMillennia);
```

Utilizzo di una funzione di conversione

Nei casi in cui la conversione è più complessa, invece di utilizzare un fattore di conversione, è possibile utilizzare una sintassi diversa e specificare una funzione che esegue la conversione. Ad esempio, è impossibile convertire valori di temperatura utilizzando un fattore di conversione, perché le varie scale termometriche hanno origini differenti.

Questo esempio, che deriva dalla unit StdConvs, mostra come registrare un tipo di conversione mediante la preparazione di funzioni per la conversione in entrambi i sensi a partire dall'unità di misura di base.

Dichiarazione delle variabili

Per prima cosa, dichiarare le variabili per gli identificativi. Gli identificativi sono utilizzati nella famiglia di conversione *cbTemperature* e nelle unità di misura che sono membri della famiglia:

```
TConvFamily cbTemperature;
TConvType tuCelsius;
TConvType tuKelvin;
TConvType tuFahrenheit;
```



Le unit di misura elencata di seguito sono un sottoinsieme delle unità di misura della temperatura effettivamente registrate nella unit StdConvs.

Registrazione della famiglia di conversione

Poi, si deve registrare la famiglia di conversione:

```
cbTemperature = RegisterConversionFamily ("Temperature");
```

Registrazione dell'unità di misura di base

Poi, definire e registrare l'unità di misura di base della famiglia di conversione, che nell'esempio sono i gradi Celsius. Notare che, nel caso dell'unità di misura di base, si può utilizzare un fattore di conversione semplice, perché non c'è alcuna conversione effettiva da fare:

```
tuCelsius = RegisterConversionType(cbTemperature,"Celsius",1);
```

Scrittura dei metodi per la conversione in base all'unità di misura di base

È necessario scrivere il codice che esegue la conversione da ogni scala termometri a gradi Celsius e viceversa, perché queste conversioni non utilizzano su un semplice fattore di conversione. Queste funzioni sono prese dalla unit StdConvs:

```
double __fastcall FahrenheitToCelsius(const double AValue)
{
  return (((AValue - 32) * 5) / 9);
}

double __fastcall CelsiusToFahrenheit(const double AValue)
{
  return (((AValue * 9) / 5) + 32);
}

double __fastcall KelvinToCelsius(const double AValue)
{
  return (AValue - 273.15);
}

double __fastcall CelsiusToKelvin(const double AValue)
{
  return (AValue + 273.15);
}
```

Registrazione delle altre unità di misura

Ora che si hanno le funzioni di conversione, è possibile registrare le altre unità di misura all'interno della famiglia di conversione. Includere anche una descrizione delle singole unità di misura.

Il codice per registrare le altre unità di misura nella famiglia è riportato di seguito:

```
tuKelvin = RegisterConversionType(cbTemperature, "Kelvin", KelvinToCelsius,
CelsiusToKelvin);
  tuFahrenheit = RegisterConversionType(cbTemperature, "Fahrenheit", FahrenheitToCelsius,
CelsiusToFahrenheit);
```

Utilizzo delle nuove unità di misura

A questo punto è possibile utilizzare le unità di misura appena registrate per eseguire le conversioni nelle applicazioni. La funzione globale *Convert* può effettuare la conversione tra tutti i tipi di conversione registrati con la famiglia di conversione *cbTemperature*. Ad esempio il codice seguente converte un valore da gradi Fahrenheit a gradi Kelvin.

```
Convert(StrToFloat(Edit1->Text), tuFahrenheit, tuKelvin);
```

Utilizzo di una classe per gestire le conversioni

È sempre possibile utilizzare funzioni di conversione per registrare una unità di conversione. A volte, però, l'operazione richiede la creazione di un numero spropositato di funzioni che fanno tutte essenzialmente la stessa cosa.

Se fosse possibile scrivere un set di funzioni di conversione che si differenziano solo per il valore di un parametro o di una variabile, sarebbe possibile creare una classe per gestire quelle conversioni. Ad esempio, dall'introduzione dell'Euro è stata approntata una serie di tecniche standard per la conversione delle diverse valute europee. Sebbene il fattore di conversione rimanga costante (a differenza del fattore di conversione, per esempio, tra dollaro e euro), è impossibile utilizzare un semplice fattore di conversione per effettuare correttamente la conversione tra valute europee per due motivi:

- La conversione deve arrotondare a un numero di cifre specifico per la valuta.
- La tecnica che si appoggia al fattore di conversione utilizza un fattore inverso rispetto a quello specificato dalle conversioni standard per l'Euro.

Tuttavia, tutto questo può essere gestito dalle funzioni di conversione come in questo caso:

```
double __fastcall FromEuro(const double AValue, const double Factor, TRoundToRange FRound)
{
  return(RoundTo(AValue * Factor, FRound));
}

double __fastcall ToEuro(const double AValue, const double Factor)
{
  return (AValue / Factor);
}
```

Il problema è che questo approccio richiede dei parametri aggiuntivi per la funzione di conversione, il che significa che è impossibile semplicemente registrare la stessa funzione per ogni valuta europea. Per evitare di dovere scrivere due nuove funzioni di conversione per ogni valuta europea, è possibile utilizzare le stesse due funzioni trasformandole in membri di una classe.

Creazione della classe di conversione

La classe deve essere un discendente di *TConvTypeFactor*. *TConvTypeFactor* definisce due metodi, *ToCommon* e *FromCommon*, per la conversione in entrambi i sensi delle unità di misura di base di una famiglia di conversione (in questo caso, da e in Euro). Come nel caso delle funzioni che si utilizzano direttamente quando si registra un'unità di conversione, questi metodi non hanno alcun parametro aggiuntivo e pertanto è necessario fornire, sotto forma di membri privati della classe di conversione, il numero di cifre per l'arrotondamento e il fattore di conversione. Lo si può vedere nell'esempio EuroConv nella directory demos\ConvertIt (vedere euroconv.pas):

```
class PASCALIMPLEMENTATION TConvTypeEuroFactor : public Convutils::TConvTypeFactor
{
  private:
```

```
    TRoundToRange FRound;
  public:
    __fastcall TConvTypeEuroFactor(const TConvFamily AConvFamily,
        const AnsiString ADescription, const double AFactor, const TRoundToRange ARound);
               TConvTypeFactor(AConvFamily, ADescription, AFactor);
    virtual double ToCommon(const double AValue);
    virtual double FromCommon(const double AValue);
}
```

Il costruttore assegna dei valori a questi membri privati:

```
__fastcall TConvTypeEuroFactor::TConvTypeEuroFactor(const TConvFamily AConvFamily,
        const AnsiString ADescription, const double AFactor, const TRoundToRange ARound):
               TConvTypeFactor(AConvFamily, ADescription, AFactor);
{
  FRound = ARound;
}
```

Le due funzioni di conversione si limitano a utilizzare questi membri privati:

```
virtual double TConvTypeEuroFactor::ToCommon(const double AValue)
{
  return (RoundTo(AValue * Factor, FRound));
}

virtual double TConvTypeEuroFactor::ToCommon(const double AValue)
{
  return (AValue / Factor);
}
```

Dichiarazione delle variabili

Ora che si ha una classe di conversione, come nel caso delle altre famiglie di conversione, si inizia con la dichiarazione degli identificatori:

```
TConvFamily cbEuro;
TConvType euEUR; // EU euro
TConvType euBEF; // Belgian francs
TConvType euDEM; // German marks
TConvType euGRD; // Greek drachmas
TConvType euESP; // Spanish pesetas
TConvType euFFR; // French francs
TConvType euIEP; // Irish pounds
TConvType euITL; // Italian lire
TConvType euLUF; // Luxembourg francs
TConvType euNLG; // Dutch guilders
TConvType euATS; // Austrian schillings
TConvType euPTE; // Protuguese escudos
TConvType euFIM; // Finnish marks
```

Registrazione della famiglia di conversione e delle altre unità

A questo punto è possibile registrare la famiglia di conversione e le unità valutarie europee, utilizzando la nuova classe di conversione. Registrare la famiglia di

conversione nello stesso modo con cui sono state registrate le altre famiglie di conversione:

```
cbEuro = RegisterConversionFamily (“European currency”);
```

Per registrare ogni tipo di conversione, creare un’istanza della classe di conversione che rifletta le proprietà del fattore di conversione e arrotondando di quella valuta e chiamare il metodo RegisterConversionType:

```
TConvTypeInfo *pInfo = new TConvTypeEuroFactor(cbEuro, “EUEuro”, 1.0, -2);
if (!RegisterConversionType(pInfo, euEUR))
  delete pInfo;
pInfo = new TConvTypeEuroFactor(cbEuro, “BelgianFrancs”, 40.3399, 0);
if (!RegisterConversionType(pInfo, euBEF))
  delete pInfo;
pInfo = new TConvTypeEuroFactor(cbEuro, “GermanMarks”, 1.95583, -2);
if (!RegisterConversionType(pInfo, euDEM))
  delete pInfo;
pInfo = new TConvTypeEuroFactor(cbEuro, “GreekDrachmas”, 340.75, 0);
if (!RegisterConversionType(pInfo, euGRD)
  delete pInfo;
pInfo = new TConvTypeEuroFactor(cbEuro, “SpanishPesetas”, 166.386, 0);
if (!RegisterConversionType(pInfo, euESP)
  delete pInfo;
pInfo = new TConvTypeEuroFactor(cbEuro, “FrenchFrancs”, 6.55957, -2);
if (!RegisterConversionType(pInfo, euFFR)
  delete pInfo;
pInfo = new TConvTypeEuroFactor(cbEuro, “IrishPounds”, 0.787564, -2);
if (!RegisterConversionType(pInfo, euIEP)
  delete pInfo;
pInfo = new TConvTypeEuroFactor(cbEuro, “ItalianLire”, 1936.27, 0);
if (!RegisterConversionType(pInfo, euITL)
  delete pInfo;
pInfo = new TConvTypeEuroFactor(cbEuro, “LuxembourgFrancs”, 40.3399, -2);
if (!RegisterConversionType(pInfo, euLUF)
  delete pInfo;
pInfo = new TConvTypeEuroFactor(cbEuro, “DutchGuilders”, 2.20371, -2);
if (!RegisterConversionType(pInfo, euNLG)
  delete pInfo;
pInfo = new TConvTypeEuroFactor(cbEuro, “AutstrianSchillings”, 13.7603, -2);
if (!RegisterConversionType(pInfo, euATS)
  delete pInfo;
pInfo = new TConvTypeEuroFactor(cbEuro, “PortugueseEscudos”, 200.482, -2);
if (!RegisterConversionType(pInfo, euPTE)
  delete pInfo;
pInfo = new TConvTypeEuroFactor(cbEuro, “FinnishMarks”, 5.94573, 0);
if (!RegisterConversionType(pInfo, euFIM)
  delete pInfo;
```

Utilizzo delle nuove unità di misura

A questo punto è possibile utilizzare le unità di misura appena registrate per eseguire le conversioni nelle applicazioni. La funzione globale *Convert* può effettuare la conversione tra tutti i tipi di valuta europei registrati con la nuova famiglia cbEuro. Ad esempio, il seguente codice converte un valore da Lire italiane a Marchi tedeschi:

```
Edit2->Text = FloatToStr(Convert(StrToFloat(Edit1->Text), euITL, euDEM));
```

Creazione di spazi di disegno

L'oggetto *TCanvas* incapsula un contesto di dispositivo Windows nella VCL, e un dispositivo di tracciamento (Qt painter) nella CLX. Gestisce tutte le operazioni di disegno per le schede, per i contenitori visuali (come i pannelli) e per l'oggetto printer (trattato in "Stampa" a pagina 4-26). Utilizzando l'oggetto canvas, voi ci si dovrà più preoccupare di istanziando penne, pennelli, tavolozze e quant'altro. L'allocazione e la deallocazione sono gestite automaticamente.

TCanvas include moltissime routine grafiche primitive per disegnare linee, figure, poligoni, caratteri, ecc. su qualsiasi controllo che contenga un canvas. Ad esempio, di seguito è riportato un gestore di evento di un pulsante che traccia una linea dall'angolo superiore sinistro al centro della scheda e scrive sulla scheda del testo non formattato:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   Canvas->Pen->Color = clBlue;
   Canvas->MoveTo( 10, 10 );
   Canvas->LineTo( 100, 100 );
   Canvas->Brush->Color = clBtnFace;
   Canvas->Font->Name = "Arial";
   Canvas->TextOut( Canvas->PenPos.x, Canvas->PenPos.y,"This is the end of the line" );
}
```

Nelle applicazioni Windows, l'oggetto *TCanvas* mette al riparo anche da comuni errori di grafica di Windows, come il ripristino di contesti di dispositivo, penne, pennelli, e così via, al valore che avevano prima dell'operazione di disegno. *TCanvas* è utilizzato in C++Builder ovunque sia necessario o possibile disegnare, e rende la grafica in Windows semplice e a prova di errore.

Per un elenco completo delle proprietà e dei metodi, consultare la Guida di riferimento in linea alla voce *TCanvas*.

Capitolo

5

Operazioni con componenti

Molti componenti sono disponibili all'interno dell'ambiente di sviluppo nella tavolozza Component. I componenti vengono selezionati dalla Component palette e rilasciati sulla scheda oppure sul modulo dati. Si progetta l'interfaccia utente dell'applicazione organizzando componenti visuali, come pulsanti e caselle di riepilogo, su una scheda. È possibile anche collocare componenti non visuali, come i componenti di accesso ai dati, o su una scheda o su un modulo dati.

Ad un primo sguardo, i componenti di C++Builder assomigliano a qualsiasi altra classe di C++. Ma ci sono delle differenze tra i componenti di C++Builder e le gerarchie standard delle classi di C++ con cui lavora la maggior parte dei programmatori C++. Eccone alcune:

- Tutti i componenti di C++Builder discendono da *TComponent*.
- I componenti nella maggioranza dei casi vengono usati così come sono e vengono modificati tramite le loro proprietà, anziché essere utilizzati come "classi base" da cui derivare sottoclassi da aggiungere o di cui modificare le funzionalità. Normalmente un componente viene ereditato per aggiungere codice specifico alle funzioni membro esistenti di gestione degli eventi.
- I componenti possono essere allocati solo nello heap, e non nello stack (*devono* cioè essere creati con l'operatore **new**).
- Le proprietà dei componenti contengono essenzialmente informazioni da utilizzare in fase di esecuzione.
- I componenti possono essere aggiunti alla Component palette nell'interfaccia utente di C++Builder e manipolati su una scheda.

Spesso i componenti raggiungono un grado di incapsulamento migliore di quello solitamente disponibile nelle classi standard di C++. Ad esempio, si consideri l'utilizzo di una finestra di dialogo che contiene un pulsante. In un programma Windows sviluppato utilizzando componenti della VCL, quando l'utente fa clic sul pulsante, il sistema genera un messaggio WM_LBUTTONDOWN. Il programma deve intercettare questo messaggio (normalmente in un'istruzione **switch**, in una

mappa di messaggi o in una tabella di risposta) e inviarlo ad una routine che viene eseguita in risposta al messaggio.

La maggior parte dei messaggi Windows (VCL) o degli eventi di sistema (CLX) sono gestiti dai componenti di C++Builder. Quando si vuole rispondere ad un messaggio, occorre solo fornire un gestore di eventi.

Il Capitolo 8, “Sviluppo dell’interfaccia utente dell’applicazione”, fornisce informazioni dettagliate sull’uso delle schede, come la creazione di schede modali in maniera dinamica, il passaggio dei parametri alle schede e il recupero dei dati dalle schede.

Impostazione delle proprietà dei componenti

Le proprietà rese pubbliche possono essere impostate in fase di progetto nell’Object Inspector e, in alcuni casi, con speciali editor di proprietà. Per impostare le proprietà in fase di esecuzione, assegnare i nuovi valori nel codice sorgente dell’applicazione.

Per informazioni sulle proprietà di ciascun componente, consultare la Guida in linea.

Impostazione delle proprietà in progettazione

Quando si seleziona un componente su una scheda in fase di progettazione, l’Object Inspector visualizza le proprietà rese pubbliche e (se possibile) consente di modificarle. Utilizzare il tasto *Tab* per passare dalla colonna Proprerty left-hand alla colonna Value right-hand. Quando il cursore è nella colonna Property, è possibile spostarsi a qualsiasi proprietà scrivendo le prime lettere del suo nome. Per proprietà di tipo Boolean o enumerativo, è possibile scegliere i valori da un elenco a discesa o commutare l’impostazione proposta facendo doppio clic sulla colonna Value.

Se accanto al nome di una proprietà appare un simbolo più (+), facendo clic su di esso, o digitando il tasto ‘+’ mentre la proprietà ha il fuoco, viene visualizzato un elenco di possibili sottovalori per la proprietà. Analogamente, se accanto al nome della proprietà viene visualizzato un segno meno (-), facendo clic sul simbolo meno, o digitando il tasto ’ -’, i sottovalori vengono nascosti.

Per impostazione predefinita, le proprietà elencate nella categoria Legacy non vengono visualizzate; per modificare i filtri di visualizzazione, fare clic destro nell’Object Inspector e scegliere il comando View. Per maggiori informazioni, consultare l’argomento “property categories” nella Guida in linea.

Nel caso siano selezionati più componenti, l’Object Inspector visualizza tutte le proprietà–tranne *Name*–che sono condivise dai componenti selezionati. Se il valore di una proprietà condivisa è diversa tra i componenti scelti, l’Object Inspector visualizza il valore predefinito oppure il valore del primo componente selezionato. Se si modifica una proprietà condivisa, la modifica viene applicata a tutti i componenti scelti.

Cambiando le proprietà relative al codice, come il nome di un gestore di eventi, nell’Object Inspector viene modificato automaticamente il corrispondente codice

sorgente. Inoltre, le modifiche apportate al codice sorgente, come la ridenominazione di un metodo gestore di eventi in una dichiarazione di classe di una scheda, si riflettono immediatamente nell'Object Inspector.

Utilizzo di editor di proprietà

Alcune proprietà, come *Font*, hanno editor di proprietà speciali. Tali proprietà mostrano un simbolo di ellissi (…). accanto ai rispettivi valori quando le si seleziona nell'Object Inspector. Per aprire l'editor di proprietà, fare doppio clic sulla colonna Value oppure fare clic sul simbolo di ellissi, oppure ancora premere la combinazione di tasti *Ctrl+Invio* quando il fuoco è sulla proprietà o sul suo valore. Con alcuni componenti, l'editor di proprietà si apre anche facendo doppio clic sul componente collocato sulla scheda.

Gli editor di proprietà consentono di impostare proprietà complesse mediante una sola finestra di dialogo. Sono in grado inoltre di convalidare i dati immessi e spesso consentono di visualizzare in anteprima il risultato di un'assegnazione.

Impostazione di proprietà in esecuzione

Ogni proprietà modificabile può essere impostata in corso di esecuzione dal codice sorgente. Per esempio, è possibile assegnare dinamicamente un titolo a una scheda:

```
Form1->Caption = MyString;
```

Chiamate a metodi

I metodi vengono chiamati come qualsiasi altra comune procedura o funzione. Per esempio, i controlli visuali hanno un metodo *Repaint* che aggiorna l'immagine del controllo sullo schermo. Il metodo *Repaint* di un oggetto draw-grid potrebbe essere chiamato nel seguente modo:

```
DrawGrid1->Repaint;
```

Come per le proprietà, il campo d'azione del un nome di metodo determina la necessità o meno di qualificatori. Se, per esempio, si desidera ridisegnare una scheda dall'interno di un gestore di evento di un controllo figlio della scheda, non è necessario anteporre il nome della scheda alla chiamata al metodo:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Repaint;
}
```

Operazioni con eventi e gestori di evento

In C++ Builder, quasi tutto il codice che si scrive viene eseguito, direttamente o indirettamente, in risposta ad *eventi*. Un evento è un tipo speciale di proprietà che rappresenta un avvenimento in esecuzione, spesso un'azione dell'utente. Il codice

che risponde direttamente a un evento–detto *gestore di evento*–è una procedura object. Le sezioni che seguono mostrano come:

- Generare un nuovo gestore di evento.
- Generare un gestore per l'evento predefinito di un componente.
- Individuare i gestori di evento.
- Associare un evento con un gestore di evento esistente.
- Associare gli eventi di menu con i gestori di evento.
- Cancellare i gestori di evento.

Generazione di nuovo gestore di evento

C++Builder è in grado di generare bozze di gestori di evento per schede ed altri componenti. Per creare un gestore di evento:

1. Selezionare un componente.
2. Fare clic sul separatore Events nell'Object Inspector. La pagina Events dell'Object Inspector visualizzerà tutti gli eventi definiti per il componente.
3. Selezionare l'evento desiderato, quindi fare doppio clic sulla colonna Value oppure premerla combinazione di tasto *Ctrl+Invio*.
4. Scrivere il codice che si desidera venga eseguito al verificarsi dell'evento.

Generazione di un gestore per l'evento predefinito di un componente

Alcuni componenti hanno un evento *predefinito*, che è l'evento che il componente dovrà gestire più spesso. Per esempio, l'evento predefinita di un pulsante è *OnClick*. Per creare un gestore di evento predefinito, fare doppio clic sul componente nel modulo di impostazione della scheda; l'operazione genera una procedura ridotta di gestione dell'evento, apre l'editor di codice e colloca il cursore nel corpo della procedura, dove si potrà aggiungere facilmente il codice necessario.

Non tutti i componenti hanno un evento predefinito. Alcuni componenti, come *TBevel*, non rispondono a nessun evento. Altri componenti rispondono in modo differente quando si fa doppio clic su di essi Form Designer. Per esempio, molti componenti, quando si fa doppio clic su di essi in fase di progettazione, aprono un editor di proprietà predefinito oppure un'altra finestra di dialogo.

Individuazione dei gestori di evento

Se per un certo componente è stato generato un gestore di evento predefinito, facendo doppio clic su di esso nel Form Designer, è possibile individuare quel gestore di evento nello stesso modo. Facendo doppio clic sul componente verrà attivato l'editor di codice e il cursore verrà collocato all'inizio del corpo del gestore di evento.

Per individuare un gestore di evento che non quello predefinito:

1 Selezionare sulla scheda il componente di cui si vuole individuare il gestore di evento.
2 Nell'Object Inspector, fare clic sul separatore Events.
3 Selezionare l'evento di cui si desidera visualizzare il gestore e fare doppio clic sulla colonna Value. Verrà aperto il Code editor con il cursore posizionato all'inizio del corpo del gestore di evento.

Associazione di un evento ad un gestore di evento esistente

Si potrebbe riutilizzare il codice scrivendo gestori di evento che rispondono a più di un evento. Ad esempio, molte applicazioni hanno pulsanti di scelta rapida che svolgono funzioni equivalenti ai comandi dei menu a comparsa. Se un pulsante attiva la stessa azione di un comando di menu, è possibile scrivere un singolo gestore di evento e assegnarlo all'evento *OnClick* sia del pulsante sia dell'elemento di menu.

Per associare un evento ad un gestore di evento esistente:

1 Sulla scheda, selezionare il componente di cui si desidera gestire l'evento.
2 Sulla pagina Events dell'Object Inspector, selezionare l'evento a cui si desidera associare un gestore.
3 Fare clic sulla freccia verso il basso nella colonna Value accanto all'evento per aprire un elenco di gestori di evento scritti in precedenza. (L'elenco include solamente i gestori di evento scritti per eventi con lo stesso nome sulla stessa scheda). Selezionare dall'elenco un gestore di evento facendo clic sul nome.

La procedura appena descritta rappresenta un modo molto semplice per riutilizzare i gestori di evento. Le *liste di azioni* , e nella VCL, le *bande di azioni*, comunque, mettono a disposizioni strumenti molto potenti per organizzare in modo centralizzato il codice che risponde ai comandi dell'utente. Le liste di azioni possono essere utilizzate in applicazioni multipiattaforma, al contrario delle bande di azioni. Per ulteriori informazioni su liste e bande di azioni, consultare "Organizzazione delle azioni per barre e menu" a pagina 8-17.

Utilizzo del parametro Sender

In un gestore di evento, il parametro *Sender* indica quale componente ha ricevuto l'evento ed ha, pertanto, chiamato il gestore. Talvolta risulta utile per far sì che diversi componenti condividano uno stesso gestore di evento che si comporta in modo differente a seconda del componente che lo ha chiamato. È possibile fare ciò utilizzando il parametro *Sender*.

Visualizzazione e codifica di eventi condivisi

Quando i componenti condividono eventi, è possibile visualizzare gli eventi condivisi nell'Object Inspector. Per prima cosa, selezionare i componenti in Form Designer tenendo premuto il tasto Maiusc e facendo clic su di essi; quindi scegliere nell'Object Inspector la pagina Events. Dalla colonna Value nell'Object Inspector, a questo punto è possibile creare un nuovo gestore di evento per alcuni degli eventi condivisi oppure assegnare ad essi un gestore di evento.

Associazione di eventi di menu a gestori di evento

Il Menu Designer, in combinazione con i componenti *MainMenu* e *PopupMenu*, semplifica l'utilizzo nell'applicazione di menu a discesa e popup. Comunque, per essere operativo, ciascun elemento di menu deve rispondere all'evento *OnClick* che si verifica ogni qualvolta l'utente sceglie l'elemento di menu oppure preme i tasti acceleratori o i tasti scorciatoia ad esso associati. Questa sezione spiega come associare i gestori di evento agli elementi di menu. Per informazioni su Menu Designer e sui componenti correlati, consultare la sezione "Creazione e gestione dei menu" a pagina 8-32.

Per creare un gestore di evento per una voce di menu:

1 Aprire il Menu Designer facendo doppio clic su un oggetto *MainMenu* o *PopupMenu*.

2 Selezionare nel Menu Designer un elemento del menu. Nell'Object Inspector, verificare che sia stato assegnato un valore alla proprietà *Name* dell'elemento.

3 Dal Menu Designer, fare doppio clic sull'elemento del menu. C++Builder genera un gestore di evento nell'editor di codice.

4 Srivere il codice che si desidera venga eseguito quando l'utente seleziona il comando del menu.

Per associare un elemento del menu con un gestore di evento *OnClick* esistente:

1 Aprire il Menu Designer facendo doppio clic su un oggetto *MainMenu* o *PopupMenu*.

2 Selezionare nel Menu Designer un elemento del menu. Nell'Object Inspector, verificare che sia stato assegnato un valore alla proprietà *Name* dell'elemento.

3 Sulla pagina Events dell'Object Inspector, fare clic sulla freccia verso il basso nella colonna Value accanto a *OnClick* per aprire un elenco dei gestori di evento scritti in precedenza. (L'elenco include solamente i gestori di evento scritti per gli eventi *OnClick* della scheda in oggetto). Effettuare una scelta nell'elenco facendo clic su un nome di gestore di evento.

Cancellazione di gestori di evento

Quando si cancella un componente da una scheda utilizzando il Form Designer, C++Builder rimuove il componente dalla dichiarazione dei tipi della scheda. Tuttavia, non cancella alcun metodo associato dal file unit, poiché questi metodi potrebbero essere chiamati ancora da altri componenti sulla scheda. È possibile cancellare manualmente un metodo,come un gestore di evento,ma, in questo caso, occorre accertarsi di cancellare sia la dichiarazione diretta del metodo sia la sua implementazione; in caso contrario, durante la generazione del progetto, si otterrà un errore del compilatore.

Componenti multipiattaforma e non multipiattaforma

La Component palette contiene una selezione di componenti in grado di gestire un'ampia gamma di operazioni di programmazione. I componenti sono disposti in diverse pagine in base al loro scopo e alle funzionalità. Per esempio, i componenti usati più frequentemente, come quelli per creare i menu, le caselle di modifica o i pulsanti, si trovano nella pagina Standard. Le pagine visualizzate nella configurazione predefinita dipendono dalla versione del prodotto che si possiede.

La Tabella 3.3 elenca le tipiche pagine predefinite e i componenti disponibili per la creazione di applicazioni, comprese quelle che non sono portabili su altre piattaforme. È possibile utilizzare tutti i componenti CLX sia in applicazioni Windows che Linux. È possibile utilizzare alcuni componenti specifici della VCL in applicazioni CLX solo per Windows, tuttavia, in tal caso, le applicazioni non saranno portabili su altre piattaforme a meno che non si isolino queste parti del codice.

Tabella 5.1 Pagine della Component palette

Nome della pagina	Descrizione	Multipiattaforma?
Standard	Controlli standard, menu	Sì
Additional	Controlli specializzati	Sì, tranne ApplicationEvents, ActionManager, ActionMainMenuBar, ActionToolBar, e CustomizeDlg. LCDNumber è solo nella CLX.
(Win32 (VCL)/ Controlli comuni (CLX)	Controlli comuni di Windows	La maggior parte dei componenti presenti sulla pagina Win32 sono anche sulla pagina Common Controls che viene visualizzata quando si crea un'applicazione CLX. RichEdit, UpDown, HotKey, Animate, DataTimePicker, MonthCalendar, Coolbar, PageScroller e ComboBoxEx si trovano solo nella VCL. TextBrowser, TextViewer, IconViewer e SpinEdit sono solo nella CLX.
System	Componenti e controlli per accesso a livello di sistema, inclusi timer, multimedia e DDE.	No, tranne Timer e PaintBox, che sono presenti sulla pagina Additional quando si crea un'applicazione CLX.
Data Access	Componenti per operazioni con dati di database che non sono legati a nessun particolare meccanismo di accesso ai dati.	Sì, tranne XMLTransform, XMLTransformProvider e XMLTransformClient.
Data Controls	Controlli visuali data-aware.	Sì, tranne DBRichEdit, DBCtrlGrid e DBChart.

Tabella 5.1 Pagine della Component palette (continua)

Nome della pagina	Descrizione	Multipiattaforma?
dbExpress	Controlli di database che utilizzano dbExpress, uno strato multipiattaforma, indipendenet dal database, che fornisce metodi per l'elaborazione SQL dinamica. Definisce un'interfaccia comune per l'accesso a server SQL.	Sì
DataSnap	Componenti utilizzati per la creazione di applicazioni database multi-tiered.	No
BDE	Componenti che forniscono l'accesso di dati tramite Borland Database Engine.	No
ADO	Componenti che forniscono l'accesso ai dati tramite il framework ADO.	No
InterBase	Componenti che forniscono accesso diretto a InterBase.	Sì
InterBaseAdmin	Componenti che accedono alle chiamate API degli InterBase Services.	Sì
InternetExpress	Componenti che sono contemporaneamente un'applicazione Web server e client di un'applicazione database multi-tiered.	No
Internet	Componenti per protocolli di comunicazione Internet e per applicazioni Web.	No
WebSnap	Componenti per la creazione di applicazioni per Web server.	No
FastNet	Controlli Internet NetMasters.	No
QReport	Componenti QuickReport per la creazione di report interni.	No
Dialogs	Finestre di dialogo di uso comune.	Sì, tranne OpenPictureDialog, SavePictureDialog, PrinterDialog e PrinterSetupDialog.
Win 3.1	Componenti in stile Win 3.1.	No
Samples	Componenti custom di esempio.	No
ActiveX	Controlli ActiveX di esempio; consultare la documentazione Microsoft (msdn.microsoft.com).	No
COM+	Componente per la gestione di eventi COM+.	No
WebServices	Componenti per la scrittura di applicazioni che implementano o utilizzano servizi Web basati su SOAP.	No
Servers	Esempi di server COM per Microsoft Excel, Word e così via (consultare la documentazione MSDN di Microsoft).	No
Indy Clients	Componenti Internet multipiattaforma per il client (componenti Internet Winshoes open-source).	Sì

Tabella 5.1 Pagine della Component palette (continua)

Nome della pagina	Descrizione	Multipiattaforma?
Indy Servers	Componenti Internet multipiattaforma per il server (componenti Internet Winshoes open-source).	Sì
Indy Misc	Componenti Internet multipiattaforma aggiuntivi (componenti Internet Winshoes open-source).	Sì

È possibile aggiungere, rimuovere e riordinare i componenti sulla tavolozza, o creare *modelli* di componenti e *frame* per raggruppare più componenti.

La Guida in linea contiene informazioni sui componenti inclusi nella Component palette. Alcuni componenti presenti nelle pagine ActiveX, Server e Samples, tuttavia, sono forniti solamente a titolo di esempio e non sono documentati.

Per ulteriori informazioni sulle differenze fra la VCL e la CLX, vedere il Capitolo 14, “Sviluppo di applicazioni multipiattaforma”.

Aggiunta di componenti custom alla Component palette

È possibile installare nella Component palette componenti custom–scritti personalmente o da terze parti– e utilizzarli nelle proprie applicazioni. Per scrivere un componente personalizzato, vedere la Parte V, “Creazione di componenti custom”. Per installare un componente esistente, consultare la sezione “Installazione di package di componenti” a pagina 15-6.

Capitolo

6

Uso dei controlli

I controlli sono componenti visuali con cui l'utente può interagire in fase di esecuzione. Questo capitolo descrive le varie caratteristiche comuni alla maggior parte dei controlli.

Implementazione del drag and drop nei controlli

La funzionalità drag-and-drop (trascinamento e rilascio) degli elementi è spesso molto utle agli utenti nella manipolazione degli oggetti. Si può fare in modo che un utente sia libero di trascinare un intero controllo, o gli elementi contenuti in un controllo—ad esempio un controllo casella di riepilogo o una vista ad albero—in un altro.

- Inizio di un'operazione di trascinamento
- Accettazione degli elementi rilasciati
- Rilascio di elementi
- Fine di un'operazione di trascinamento
- Personalizzazione delle operazioni di drag-and-drop con un oggetto drag
- Modifica del puntatore di trascinamento del mouse

Inizio di un'operazione di trascinamento

Ogni controllo ha una proprietà di nome *DragMode* che determina come vengono iniziate le operazioni di trascinamento. Se *DragMode* è *dmAutomatic*, il trascinamento inizia automaticamente quando l'utente preme un pulsante del mouse con il cursore sul controllo. Dal momento che *dmAutomatic* può interferire con la normale attività del mouse, è opportuno impostare *DragMode* a *dmManual* (impostazione predefinita) e iniziare il trascinamento per gestire gli eventi mouse-down.

Per iniziare a trascinare manualmente un controllo, occorre chiamare il suo metodo *BeginDrag*. *BeginDrag* accetta un parametro booleano di nome *Immediate* e un

parametro integer di nome *Threshold*. Se si passa **true** per *Immediate*, il trascinamento inizia immediatamente. Se si passa **false**, il trascinamento non inizia finché l'utente non sposta il mouse del numero di pixel specificati da *Threshold*. Se il valore di *Threshold* è -1, viene utilizzato un valore predefinito. La chiamata

```
BeginDrag (false, -1);
```

consente al controllo di accettare i clic del mouse senza iniziare un'operazione di trascinamento.

Verificando i parametri dell'evento mouse-down prima di chiamare *BeginDrag*, è possibile porre altre condizioni su quando iniziare il trascinamento, come il controllo del pulsante che ha premuto l'utente. Il codice seguente, per esempio, gestisce un evento mouse-down in una casella di riepilogo di file iniziando il trascinamento solo se è stato premuto il pulsante sinistro del mouse.

```
void __fastcall TFMForm::FileListBox1MouseDown(TObject *Sender,
  TMouseButton Button, TShiftState Shift, int X, int Y)
{
  if (Button == mbLeft)// drag only if left button pressed
  {
    TFileListBox *pLB = (TFileListBox *)Sender; // cast to TFileListBox
    if (pLB->ItemAtPos(Point(X,Y), true) >= 0) // is there an item here?
      pLB->BeginDrag(false, -1);                  // if so, drag it
  }
}
```

Accettazione degli elementi rilasciati

Quando l'utente trascina qualcosa su un controllo, questo riceve un evento *OnDragOver*, e in quel momento deve indicare se può accettare l'elemento nel caso in cui l'utente dovesse rilasciarlo. Il cursore di trascinamento si trasforma per indicare se il controllo può accettare o meno l'elemento trascinato. Per accettare gli elementi trascinati su un controllo, occorre collegare il gestore all'evento *OnDragOver* del controllo.

L'evento drag-over ha un parametro di nome *Accept* che il gestore di evento può impostare a **true** se accetterà l'elemento. *Accept* può modificare o meno il tipo di cursore in un cursore di accettazione.

L'evento drag-over possiede altri parametri, compresa la sorgente del trascinamento e la posizione corrente del cursore del mouse, che il gestore di evento può usare per determinare se accettare o meno il rilascio. Nell'esempio seguente una vista dell'albero della directory accetta gli elementi trascinati soltanto se provengono da una casella di riepilogo di file.

```
void __fastcall TForm1::TreeView1DragOver(TObject *Sender, TObject *Source,
  int X, int Y, TDragState State, bool &Accept)
{
  if (Source->InheritsFrom(__classid(TFileListBox)))
    Accept = true;
}
```

Rilascio di elementi

Se un controllo indica che può accettare un elemento trascinato, è necessario gestire l'elemento che viene rilasciato su di esso. Per gestire gli elementi rilasciati, si deve collegare un gestore all'evento *OnDragDrop* del controllo che accetta l'elemento rilasciato. Come l'evento drag-over, l'evento drag-and-drop indica la sorgente dell'elemento trascinato e le coordinate del cursore del mouse sul controllo accettante. L'ultimo parametro consente di monitorare il percorso compiuto da un oggetto durante il trascinamento; ad esempio, si potrebbe utilizzare questa informazione per modificare il colore dei componenti, man mano che l'oggetto passa sopra di essi.

Nell'esempio che segue, una vista ad albero della directory,che accetta elementi trascinati da una casella di riepilogo di file, risponde spostando i file nella directory su cui questi vengono rilasciati.

```
void __fastcall TForm1::TreeView1DragDrop(TObject *Sender, TObject *Source,
int X, int Y){
  if (Source->InheritsFrom(__classid(TFileListBox)))
  {
    TTreeNode *pNode = TreeView1->GetNodeAt(X,Y); // pNode is drop target
    AnsiString NewFile = pNode->Text + AnsiString("//") +
      ExtractFileName(FileListBox1->FileName); // build file name for drop target
    MoveFileEx(FileListBox1->FileName.c_str(), NewFile.c_str(),
      MOVEFILE_REPLACE_EXISTING | MOVEFILE_COPY_ALLOWED); // move the file
  }
}
```

Fine di un'operazione di trascinamento

Un'operazione di trascinamento termina o quando l'elemento viene rilasciato con successo oppure quando viene rilasciato su un controllo che non è in grado di accettarlo. A questo punto, al controllo da cui l'elemento è stato trascinato, viene inviato un evento end-drag. Affinché un controllo possa rispondere quando gli si prelevano degli elementi, occorre associare all'evento *OnEndDrag* del controllo un gestore di evento.

Il parametro più importante di un evento *OnEndDrag* è chiamato *Target*, per indicare qual è il controllo eventuale che accetta il rilascio. Se *Target* è null, significa che nessun controllo accetta l'elemento trascinato. L'evento *OnEndDrag* include le coordinate sul controllo ricevente.

In questo esempio, una casella di riepilogo di file gestisce un evento end-drag effettuando l'aggiornamento del suo elenco di file.

```
void __fastcall TFMForm::FileListBox1EndDrag(TObject *Sender, TObject *Target,  int X, int
Y)
  if (Target)
    FileListBox1->Update();
};
```

Personalizzazione delle operazioni di drag-and-drop con un oggetto drag

Per personalizzare il comportamento drag-and-drop dell'oggetto è possibile usare un discendente di *TDragObject*. Per impostazione predefinita, gli eventi drag-over e drag-and-drop indicano la sorgente dell'elemento trascinato e le coordinate del cursore del mouse sul controllo accettante. Per ottenere altre informazioni dello stato, derivare da *TDragObject* o da *TDragObjectEx*(solo per VCL) un oggetto drag custom e ridefinirne i metodi virtuali. Creare l'oggetto drag custom nell'evento *OnStartDrag*.

Di solito, il parametro sorgente degli eventi drag-over e drag-and-drop è il controllo che inizia l'operazione di trascinamento. Se un'operazione che coinvolge gli stessi tipi di dati può essere iniziata da tipi di controllo differenti, il sorgente deve supportare ciascun tipo di controllo. Comunque, se si utilizza un discendente di *TDragObject*, il sorgente è lo stesso oggetto drag; se ogni controllo crea lo stesso tipo di oggetto drag nel proprio evento *OnStartDrag*, il controllo di destinazione deve essere in grado di gestire un solo tipo di oggetto. Gli eventi drag-over e drag-and-drop possono riferire se il sorgente è un oggetto drag, invece del controllo, chiamando la funzione *IsDragObject*.

I discendenti di *TDragObjectEx* (solo per VCL) vengono liberati automaticamente al contrario dei discendenti di *TDragObject*. Se si hanno discendenti di *TDragObject* che non vengono liberati esplicitamente, è possibile modificarli in modo che discendano invece da *TDragObjectEx* in modo da evitare perdite di memoria.

Gli oggetti drag consentono di trascinare elementi fra una scheda implementata nel file EXE principale dell'applicazione e una scheda implementata utilizzando una DLL, o fra schede implementate utilizzando DLL differenti.

Modifica del puntatore di trascinamento del mouse

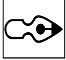
Impostando la proprietà *DragCursor* del componente sorgente è possibile personalizzare l'aspetto del puntatore del mouse durante le operazioni di trascinamento).

Implementazione del drag-and-dock nei controlli

I discendenti di *TWinControl* possono agire come posizioni di aggancio e i discendenti di *TControl* possono, invece, comportarsi come finestre figlio che vengono agganciate alle posizioni specificate. Per esempio, per fornire una posizione di aggancio al margine sinistro di una finestra scheda, si deve allineare un riquadro al margine sinistro della scheda e renderlo una posizione di aggancio. Quando i controlli agganciabili vengono trascinati nel riquadro e vengono rilasciati, diventano controlli figlio del riquadro.

- Trasformazione di un controllo con finestra in una posizione di aggancio
- Trasformazione di un controllo in un figlio agganciabile
- Controllo della modalità di aggancio dei controlli figlio

- Controllo della modalità di sgancio dei controlli figlio
- Controllo della modalità di risposta dei controlli figlio alle operazioni drag-and-dock



Le proprietà drag-and-dock sono disponibili nella VCL ma non in CLX.

Trasformazione di un controllo con finestra in una posizione di aggancio

Per rendere un controllo finestra una posizione di aggancio:

1 Impostare la proprietà *DockSite* a **true**.

2 Se l'oggetto della posizione di aggancio non deve apparire se non contiene un client agganciato, impostare la sua proprietà *AutoSize* a **true**. Quando *AutoSize* è **true**, la posizione di aggancio ha come dimensione 0 finché non accetta un controllo figlio per l'aggancio. Quindi, viene ridimensionata in modo tra trovare posto intorno al controllo figlio.

Trasformazione di un controllo in un figlio agganciabile

Per trasformare un controllo in un figlio agganciabile:

1 Impostare la sua proprietà *DragKind* a *dkDock*. Quando *DragKind* è *dkDock*, il trascinamento del controllo lo sposta in una nuova posizione di aggancio o sgancia il controllo in modo che diventi una finestra mobile. Quando *DragKind* è *dkDrag* (impostazione predefinita), il trascinamento del controllo avvia un'operazione drag-and-drop che deve essere implementata usando gli eventi *OnDragOver*, *OnEndDrag* e *OnDragDrop*.

2 Impostare la sua proprietà *DragMode* a *dmAutomatic*. Quando *DragMode* è *dmAutomatic*, il trascinamento (per l'operazione drag-and-drop o di aggancio, in funzione di *DragKind*) viene iniziato automaticamente quando l'utente avvia il trascinamento del controllo con il mouse. Quando *DragMode* è *dmManual*, è ancora possibile iniziare un'operazione drag-and-dock (o drag-and-drop) chiamando il metodo *BeginDrag*.

3 Impostare la sua proprietà *FloatingDockSiteClass* per indicare il discendente *TWinControl* che ospiterà il controllo quando sarà sganciato e lasciato come finestra mobile. Quando il controllo viene rilasciato in una posizione diversa da quella di aggancio, viene creato dinamicamente un controllo finestra di questa classe, che diventa il genitore del figlio agganciabile. Se il controllo del figlio agganciabile è un discendente di *TWinControl*, non è necessario creare una distinta posizione di aggancio mobile per ospitare il controllo, anche se può essere opportuno specificare una scheda per ottenere un bordo e una barra del titolo. Per omettere una finestra contenitore dinamica, si deve impostare *FloatingDockSiteClass* alla stessa classe del controllo; in questo modo diventerà una finestra mobile senza alcun genitore.

Controllo della modalità di aggancio dei controlli figlio

Un posizione di aggancio accetta automaticamente i controlli figlio quando vi vengono rilasciati sopra. Per la maggior parte dei controlli il primo figlio viene agganciato per riempire l'area client, il secondo la divide in due regioni distinte, e così via. I controlli pagina agganciano i figli in nuovi fogli di tabulazione (o fondono i fogli di tabulazione se il figlio è un altro controllo pagina).

Tre eventi consentono alle posizioni di aggancio di limitare ulteriormente il modo in cui vengono agganciati i controlli figlio:

```
__property TGetSiteInfoEvent OnGetSiteInfo = {read=FOnGetSiteInfo, write=FOnGetSiteInfo};
typedef void __fastcall  (__closure *TGetSiteInfoEvent)(System::TObject* Sender, TControl*
DockClient, Windows::TRect &InfluenceRect, const Windows::TPoint &MousePos, bool &CanDock);
```

OnGetSiteInfo si verifica nella posizione di aggancio quando l'utente trascina un figlio agganciabile sul controllo. Questo evento consente alla posizione di indicare se accetterà o meno il controllo specificato dal parametro *DockClient* come figlio, e in caso di risposta positiva, dove deve essere agganciato il figlio. Quando si verifica *OnGetSiteInfo*, *InfluenceRect* viene inizializzato alle coordinate dello schermo della posizione di aggancio, e *CanDock* viene inizializzato a **true**. Modificando *InfluenceRect*, può essere creata una regione di aggancio più limitata, mentre il figlio può essere rifiutato impostando *CanDock* a **false**.

```
__property TDockOverEvent OnDockOver = {read=FOnDockOver, write=FOnDockOver};
typedef void __fastcall  (__closure *TDockOverEvent)(System::TObject* Sender,
TDragDockObject* Source, int X, int Y, TDragState State, bool &Accept);
```

OnDockOver si verifica nella posizione di aggancio quando l'utente trascina sul controllo un figlio agganciabile. Corrisponde all'evento *OnDragOver* in una normale operazione drag-and-drop. Deve essere utilizzato per segnalare che il figlio può essere rilasciato dalla posizione di aggancio, impostando il parametro *Accept*. Se il controllo agganciabile è stato rifiutato dal gestore di evento *OnGetSiteInfo* (forse perché il tipo di controllo era errato), l'evento *OnDockOver* non verrà attivato.

```
__property TDockDropEvent OnDockDrop = {read=FOnDockDrop, write=FOnDockDrop};
typedef void __fastcall  (__closure *TDockDropEvent)(System::TObject* Sender,
TDragDockObject* Source, int X, int Y);
```

OnDockDrop si verifica nella posizione di aggancio quando l'utente rilascia un figlio agganciabile sul controllo. Corrisponde all'evento *OnDragDrop* in una normale operazione drag-and-drop. Deve essere utilizzato per eseguire un adattamento necessario per accettare il controllo come controllo figlio. L'accesso al controllo figlio può essere ottenuto usando la proprietà *Control* del *TDockObject* specificato dal parametro *Source*.

Controllo della modalità di sgancio dei controlli figlio

Una posizione di controllo consente automaticamente ai controlli figlio di essere sganciati quando vengono trascinati ed hanno la proprietà *DragMode* impostata a *dmAutomatic*. Le posizioni di aggancio possono rispondere quando i controlli figlio

vengono trascinati fuori, e possono anche impedire lo sganciamento in un gestore di evento *OnUnDock*:

```
__property TUnDockEvent OnUnDock = {read=FOnUnDock, write=FOnUnDock};
typedef void __fastcall  (__closure *TUnDockEvent)(System::TObject* Sender, TControl*
Client, TWinControl* NewTarget, bool &Allow);
```

Il parametro *Client* indica il controllo figlio che sta tentando lo sganciamento, mentre il parametro *Allow* consente alla posizione di aggancio (*Sender*) di rifiutare lo sganciamento. Quando si implementa un gestore di evento *OnUnDock*, può risultare utile conoscere quali altri figli (eventuali) sono al momento agganciati. Queste informazioni sono disponibili nella proprietà a sola lettura *DockClients*, che è un array indicizzato di *TControl*. Il numero di client agganciati viene indicato dalla proprietà a sola lettura *DockClientCount*.

Controllo della modalità di risposta dei controlli figlio alle operazioni drag-and-dock

I controlli figlio agganciabili hanno due eventi che si verificano durante le operazioni drag-and-dock: *OnStartDock*, analogo all'evento *OnStartDrag* di un'operazione drag-and-drop, consente al controllo figlio agganciabile di creare un oggetto trascinamento personalizzato. *OnEndDock*, analogamente all'evento *OnEndDrag*, si verifica al termine del trascinamento.

Operazioni con testo nei controlli

Le sezioni seguenti spiegano come utilizzare le varie funzioni dei controlli RichEdit e Memo. Alcune funzioni qui descritte valgono anche per i controlli di editing.

- Impostazione dell'allineamento del testo
- Aggiunta di barre di scorrimento in fase di esecuzione
- Aggiunta dell'oggetto clipboard
- Selezione di testo
- Selezione di tutto il testo
- Operazioni taglia, copia e incolla del testo
- Cancellazione del testo selezionato
- Disattivazione delle voci di menu
- Preparazione di un menu a comparsa
- Gestione dell'evento OnPopup

Impostazione dell'allineamento del testo

In un componente RichEdit o Memo, il testo può essere allineato a sinistra, a destra o al centro. Per cambiare l'allineamento del testo, impostare la proprietà *Alignment* del componente di editing. L'allineamento ha effetto solo se la proprietà *WordWrap* è **true**; se l'allineamento alla riga successiva è disattivato, non esiste alcun margine rispetto al quale effettuare l'allineamento.

Ad esempio, il codice seguente tratto dall'esempio RichEdit imposta l'allineamento a seconda del pulsante scelto:

```
switch((int)RichEdit1->Paragraph->Alignment)
{
  case 0: LeftAlign->Down   = true; break;
  case 1: RightAlign->Down  = true; break;
  case 2: CenterAlign->Down = true; break;
}
```

Aggiunta di barre di scorrimento in fase di esecuzione

I componenti RichEdit e Memo possono contenere barre di scorrimento orizzontali o verticali, o entrambe, a seconda delle necessità. Quando l'allineamento alla riga successiva è attivato, il componente ha bisogno solo di una barra di scorrimento verticale. Se l'utente disattiva l'allineamento alla riga successiva, il componente può richiedere anche una barra di scorrimento orizzontale, poiché il testo non è delimitato dal lato destro dell'editor.

Aggiunta di barre di scorrimento in fase di esecuzione:

1 Determinare se il testo può superare o meno il margine destro. In molti casi, ciò significa controllare se la funzionalità di a capo automatico è abilitata. Si può anche controllare se qualche riga di testo supera la larghezza del controllo.

2 Impostare la proprietà *ScrollBars* del componente RichEdit o Memo per includere o escludere le barre di scorrimento.

L'esempio seguente collega un gestore di evento *OnClick* alla voce di menu Character|WordWrap.

```
void __fastcall TEditForm::WordWrap1Click(TObject *Sender)
{
  Editor->WordWrap = !(Editor->WordWrap);   // toggle word wrapping
  if (Editor->WordWrap)
    Editor->ScrollBars = ssVertical;        // wrapped requires only vertical
  else
    Editor->ScrollBars = ssBoth;            // unwrapped can need both
  WordWrap1->Checked = Editor->WordWrap;    // check menu item to match property
}
```

I componenti RichEdit e Memo gestiscono le loro barre di scorrimento in modo leggermente diverso. Il componente RichEdit può nascondere le sue barre di scorrimento se il testo sta all'interno dei suoi margini. Il componente Memo mostra sempre le barre di scorrimento se sono attivate.

Aggiunta dell'oggetto clipboard

La maggior parte delle applicazioni di gestione del testo fornisce agli utenti un modo per spostare il testo selezionato tra i documenti, compresi quelli di applicazioni differenti. L'oggetto *Clipboard* in C++Builder incapsula una clipboard (come la clipboard di Windows) e include i metodi per tagliare, copiare e incollare il testo (e

altri formati, compresi i grafici). L'oggetto *Clipboard* viene dichiarato nella unit *Clipbrd*.

Aggiunta dell'oggetto *Clipboard* ad un'applicazione:

1 Selezionare la unit che userà la clipboard.
2 Nel file .h della scheda, aggiungere

```
#include <vcl\Clipbrd.hpp>
```

Selezione del testo

Prima di poterlo inviare nella clipboard, il testo desiderato deve essere selezionato. L'evidenziazione del testo selezionato è incorporata nei componenti Edit. Quando l'utente seleziona il testo, esso appare evidenziato.

La Tabella 6.1 elenca le proprietà usate comunemente per gestire il testo selezionato.

Tabella 6.1 Proprietà del testo selezionato

Proprietà	Descrizione
SelText	Contiene una stringa che rappresenta il testo selezionato nel componente.
SelLength	Contiene la lunghezza di una stringa selezionata.
SelStart	Contiene la posizione iniziale di una stringa relativa all'inizio di un buffer di testo di un controllo edit.

Selezione di tutto il testo

Il metodo *SelectAll* seleziona tutto il contenuto di un controllo di editing, come ad esempio, un componente rich edit o memo. Ciò è particolarmente utile quando il contenuto del componente supera l'area visibile del componente. In molti altri casi gli utenti selezionano il testo con le combinazioni di tasti o con il trascinamento del mouse.

Per selezionare l'intero contenuto di un controllo RichEdit o Memo, si deve chiamare il metodo *SelectAll* del controllo *RichEdit1*.

Ad esempio:

```
void __fastcall TMainForm::SelectAll(TObject *Sender)
{
  RichEdit1->SelectAll();     // select all text in RichEdit
}
```

Operazione taglia, copia e incolla del testo

Le applicazioni che usano la unit *Clipbrd* possono tagliare, copiare e incollare testo, grafici e oggetti tramite la clipboard. Tutti i componenti Edit che incapsulano i controlli standard di Windows per la gestione del testo hanno metodi incorporati per

interagire con la clipboard. (Per informazioni sull'uso della clipboard con i grafici, consultare il paragrafo "Uso della clipboard con i grafici" a pagina 10-22.)

Per tagliare, copiare o incollare testo tramite la clipboard, si devono chiamare, rispettivamente, i metodi *CutToClipboard*, *CopyToClipboard*, e *PasteFromClipboard* del componente Edit.

Per esempio, il codice seguente collega i gestori agli eventi *OnClick*, rispettivamente dei comandi Edit|Cut, Edit|Copy e Edit|Paste:

```
void __fastcall TMainForm::EditCutClick(TObject* Sender)
{    RichEdit1->CutToClipboard();
}
void __fastcall TMainForm::EditCopyClick(TObject* Sender)
{    RichEdit1->CopyToClipboard();
}
void __fastcall TMainForm::EditPasteClick(TObject* Sender)
{    RichEdit1->PasteFromClipboard();
}
```

Cancellazione del testo selezionato

È possibile cancellare il testo selezionato in un componente Edit senza tagliarlo e collocarlo nella clipboard. Per fare ciò, chiamare il metodo *ClearSelection*. Ad esempio, se nel menu Edit c'è un elemento Delete, il codice potrebbe assomigliare al seguente:

```
void __fastcall TMainForm::EditDeleteClick(TObject *Sender)
{
  RichEdit1->ClearSelection();
}
```

Disattivazione delle voci di menu

È spesso utile disattivare i comandi di menu senza rimuoverli dal menu. Per esempio, in un editor di testi, se al momento non c'è alcun testo selezionato, non è possibile utilizzare i comandi Cut, Copy e Delete. L'attivazione o la disattivazione delle voci di menu può essere compiuta, ad esempio, nel momento in cui l'utente seleziona il menu. Per disattivare una voce di menu, impostarne la proprietà *Enabled* a **false**.

Nell' esempio seguente, un gestore viene collegato all'evento *OnClick* della voce Edit in una barra di menu della scheda figlio. Questo gestore imposta la proprietà *Enabled* delle voci Cut, Copy e Delete del menu Edit a seconda che in *RichEdit1* vi sia o meno del testo selezionato. Il comando Paste viene attivato o disattivato in base al fatto che esista o meno un testo nella clipboard.

```
void __fastcall TMainForm::EditEditClick(TObject *Sender)
{
  // enable or disable the Paste menu item
  Paste1->Enabled = Clipboard()->HasFormat(CF_TEXT);
  bool HasSelection = (RichEdit1->SelLength > 0);  // true if text is selected
```

```
    Cut1->Enabled = HasSelection;  // enable menu items if HasSelection is true
    Copy1->Enabled = HasSelection;
    Delete1->Enabled = HasSelection;
  }
```

Il metodo *HasFormat* della clipboard restituisce un valore booleano a seconda che la clipboard contenga o meno oggetti, testo o immagini di un particolare formato. Chiamando *HasFormat* con il parametro *CF_TEXT*, si può determinare se la clipboard contenga o meno un testo, e attivare o disattivare la voce Paste.

Il Capitolo 10, "Operazioni con elementi grafici e multimediali" fornisce ulteriori informazioni sull'utilizzo della clipboard con i grafici.

Preparazione di un menu a comparsa

I menu a comparsa, o locali, sono una caratteristica comune facile da usare per qualsiasi applicazione. Essi consentono agli utenti di ridurre al minimo lo spostamento del mouse facendo clic sul pulsante destro nello spazio di lavoro dell'applicazione per accedere ad un elenco di comandi di uso frequente.

In un'applicazione text editor, per esempio, si può aggiungere un menu a comparsa che ripeta i comandi di editing Cut, Copy e Paste. Queste voci del menu a comparsa possono usare gli stessi gestori di evento delle corrispondenti voci del menu Edit. Non occorre creare combinazioni di tasti o tasti di scelta rapida per i menu a comparsa, perché le voci corrispondenti del menu corrispondenti generalmente hanno già dei tasti di scelta rapida.

La proprietà *PopupMenu* di una scheda specifica quale menu a comparsa visualizzare quando un utente fa clic con il tasto destro su un elemento della scheda. I singoli controlli hanno anche proprietà *PopupMenu* che possono ridefinire la proprietà della scheda, consentendo l'uso di menu personalizzati per particolari controlli.

Per aggiungere un menu a comparsa a una scheda:

1 Collocare un componente *PopupMenu* sulla scheda.
2 Usare il Menu Designer per definire le voci del menu a comparsa.
3 Impostare la proprietà *PopupMenu* della scheda o del controllo che visualizza il menu al nome del componente menu a comparsa.
4 Collegare i gestori agli eventi *OnClick* delle voci del menu a comparsa.

Gestione dell'evento OnPopup

Può essere opportuno regolare le voci del menu a comparsa prima di visualizzare il menu, proprio come avviene per l'attivazione o disattivazione delle voci di un normale menu. Con un menu normale è possibile gestire l'evento *OnClick* per la voce posta all'inizio del menu, come descritto nel paragrafo "Disattivazione delle voci di menu" a pagina 6-10.

Con un menu a comparsa, invece, non c'è alcuna barra di menu al livello più alto, quindi per preparare i comandi del menu a comparsa occorre gestire l'evento nel

componente menu stesso. Il componente menu a comparsa fornisce un evento, chiamato, solo per questo scopo, *OnPopup*.

Per regolare le voci su un menu a comparsa prima di visualizzarle:

1. Selezionare il componente menu a comparsa.
2. Collegare un gestore al suo evento *OnPopup*.
3. Scrivere il codice nel gestore di evento per attivare, disattivare, nascondere o mostrare le voci del menu.

Nel seguente codice il gestore di evento *Edit1Click*, descritto precedentemente in "Disattivazione delle voci di menu" a pagina 6-10 viene collegato all'evento *OnPopup* del componente menu a comparsa. A *Edit1Click* viene aggiunta una riga di codice per ciascuna voce del menu popup.

```
void __fastcall TMainForm::EditEditClick(TObject *Sender)
{
  // enable or disable the Paste menu item
  Paste1->Enabled = Clipboard()->HasFormat(CF_TEXT);
  Paste2->Enabled = Paste1->Enabled;           // add this line
  bool HasSelection = (RichEdit1->SelLength > 0);  // true if text is selected
  Cut1->Enabled = HasSelection;    // enable menu items if HasSelection is true
  Cut2->Enabled = HasSelection;    // add this line
  Copy1->Enabled = HasSelection;
  Copy2->Enabled = HasSelection;   // add this line
  Delete1->Enabled = HasSelection;
}
```

Aggiunta di grafici ai controlli

Diversi controlli consentono di personalizzare il modo in cui il controllo viene rappresentato. Questi controlli includono caselle di riepilogo, caselle combinate, menu, headers, controlli multipagina, viste a elenco, barre di stato, viste ad albero e barre strumenti. Fra questi vi sono caselle di riepilogo, caselle combinate, menu, intestazioni, controlli pagina, viste elenco, barre di stato, viste ad albero e barre strumenti. L'uso più comune dei controlli owner-draw consiste nel fornire grafici in sostituzione o in aggiunta al testo per gli elementi. Per informazioni su come usare lo stile owner-draw per aggiungere immagini ai menu, consultare "Aggiunta di immagini alle voci di menu" a pagina 8-38.

Tutti i controlli owner-draw contengono elenchi di elementi.Di solito, quelle liste sono liste di stringhe che vengono visualizzate come testo, oppure liste di oggetti che contengono stringhe che vengono visualizzate come testo. È possibile associare un oggetto a ciascun elemento di un elenco per facilitarne l'uso quando si disegnano gli elementi.

In generale, la creazione di un controllo owner-draw in C++Builder implica la conoscenza dei seguenti argomenti:

1. Come indicare che un controllo è owner-drawn.
2. Aggiunta di oggetti grafici ad un elenco di stringhe.
3. Disegno di elementi owner-draw

Come indicare che un controllo è owner-drawn

Per personalizzare il tracciamento di un controllo, è necessario approntare dei gestori di evento che siano in grado di tracciare all'occorrenza l'immagine del controllo.Alcuni controlli ricevono questi eventi in modo automatico. Ad esempio, viste elenco, viste ad albero e barre strumenti ricevono tutti eventi a diversi stadi durante il processo di tracciamento, senza dover impostare alcuna proprietà. Questi eventi hanno nomi come "OnCustomDraw" oppure "OnAdvancedCustomDraw".

Altri controlli, tuttavia, necessitano che l'impostazione una proprietà prima che possano ricevare eventi owner-draw. Le caselle di riepilogo, le caselle combinate, i controlli header e le barre di stato hanno una proprietà di nome *Style*. *Style* determina se il controllo utilizza o meno il disegno predefinito (chiamato stile "standard") o il disegno personalizzato. Le griglie utilizzano una proprietà di nome *DefaultDrawing* per attivare o disattivare il disegno predefinito. Le viste ad elenco e i controlli multipagina hanno una proprietà di nome *OwnerDraw* che attiva o disattiva il disegno predefinito.

Le caselle di riepilogo e le caselle combinate hanno stili owner-draw aggiuntivi, chiamati *fixed* e *variable*, come descritto nella Tabella 6.2. Altri controlli sempre sono *fixed* e, sebbene la dimensione dell'elemento che contiene il testo può variare, la dimensione di ciascuno elemento viene determinata prima di disegnare il controllo.

Tabella 6.2 Confronto dello stile owner-draw fixed con quello variable

Stile owner-draw	Significato	Esempi
Fixed	Ogni elemento ha la stessa altezza, e precisamente quella determinata dalla proprietà *ItemHeight*.	*lbOwnerDrawFixed*, *csOwnerDrawFixed*
Variable	Ogni elemento può avere un'altezza diversa, determinata dai dati forniti al momento dell'esecuzione.	*lbOwnerDrawVariable*, *csOwnerDrawVariable*

Aggiunta di oggetti grafici ad un elenco di stringhe

Ogni elenco di stringhe ha la possibilità di contenere un elenco di oggetti, in aggiunta al suo elenco di stringhe.

Per esempio, in un'applicazione file manager, può essere opportuno aggiungere le bitmap che indicano il tipo di unità insieme alla relativa lettera. A questo scopo, occorre aggiungere le immagini bitmap all'applicazione, quindi copiarle nei posti corretti dell'elenco di stringhe, come descritto nelle sezioni seguenti.

Aggiunta di immagini ad un'applicazione

Un controllo Image è un controllo non visuale che contiene un'immagine grafica, come una bitmap. I controlli immagine vengono usati per visualizzare immagini grafiche su una scheda. Inoltre, possono essere usati per contenere immagini nascoste che verranno usate nell'applicazione. Per esempio, si possono memorizzare bitmap per controlli owner-draw in controlli Image nascosti nel modo seguente:

1 Aggiungere i controlli immagine alla scheda principale.
2 Impostare le loro proprietà *Name*.
3 Impostare la proprietà *Visible* per ciascun controllo immagine a **false**.
4 Impostare la proprietà *Picture* di ciascuna immagine alla bitmap desiderata tramite l'editor *Picture* dall'Object Inspector.

I controlli immagine sono invisibili quando viene eseguita l'applicazione.

Aggiunta di immagini a un elenco di stringhe

Dopo aver aggiunto immagini grafiche ad un'applicazione, è possibile associarle alle stringhe di un elenco. È possibile aggiungere gli oggetti tutti in un volta, come le stringhe, o associare gli oggetti alle stringhe esistenti. Il metodo preferito è quello di aggiungere oggetti e stringhe contemporaneamente, se si hanno a disposizione tutti i dati necessari.

L'esempio seguente mostra come si possono aggiungere immagini ad un elenco di stringhe. L'esempio fa parte di un'applicazione file manager in cui, insieme ad una lettera per ciascuna unità valida, viene aggiunto una bitmap che indica il tipo di ciascuna unità. Il gestore di evento *OnCreate* ha questo formato:

```
void __fastcall TFMForm::FormCreate(TObject *Sender)
{
  int AddedIndex;
  char DriveName[4] = "A:\\";
  for (char Drive = 'A'; Drive <= 'Z'; Drive++) // try all possible drives
  {
    DriveName[0] = Drive;
    switch (GetDriveType(DriveName))
    {
      case DRIVE_REMOVABLE:// add a list item
        DriveName[1] = '\0'; // temporarily make drive letter into string
        AddedIndex = DriveList->Items->AddObject(DriveName,
           Floppy->Picture->Graphic);
        DriveName[1] = ':' // replace the colon
        break;
      case DRIVE_FIXED:// add a list item
        DriveName[1] = '\0'; // temporarily make drive letter into string
        AddedIndex = DriveList->Items->AddObject(DriveName,
           Fixed->Picture->Graphic);
        DriveName[1] = ':' // replace the colon
        break;
      case DRIVE_REMOTE:// add a list item
        DriveName[1] = '\0'; // temporarily make drive letter into string
        AddedIndex = DriveList->Items->AddObject(DriveName,
           Network->Picture->Graphic);
        DriveName[1] = ':' // replace the colon
        break;
    }
    if ((int)(Drive - 'A') == getdisk()) // current drive?
      DriveList->ItemIndex = AddedIndex; // then make that the current list item
  }
}
```

Disegno di elementi owner-draw

Le viste elenco e i controlli pagina hanno una proprietà di nome *OwnerDraw* che attiva o disattiva il tracciamento predefinito. Invece, il sistema operativo genera eventi per ciascun elemento visibile del controllo. L'applicazione gestisce gli eventi per disegnare gli elementi.

Per disegnare gli elementi in un controllo owner-draw, occorre fare quanto descritto qui di seguito per ciascun elemento visibile del controllo. Si deve usare un unico gestore di evento per tutti gli elementi.

1. Dimensionare, se necessario, l'elemento.

   Gli elementi con le stesse dimensioni (per esempio, con uno stile casella di riepilogo di *lsOwnerDrawFixed*), non richiedono alcun dimensionamento.

2. Disegno dell'elemento.

Dimensionamento degli elementi owner-draw

Prima di dare all'applicazione la possibilità di disegnare tutti gli elementi di un controllo owner-draw variabile, il sistema operativo genera un evento a misura di elemento. Questo evento comunica all'applicazione dove l'elemento appare sul controllo.

C++Builder determina le dimensioni dell'elemento (generalmente, sono grandi a sufficienza per visualizzare il testo dell'elemento con il font corrente). L'applicazione può gestire l'evento e modificare il rettangolo scelto. Per esempio, se si prevede di sostituire una bitmap per il testo dell'elemento, è bene modificare il rettangolo in modo che abbia le dimensioni del bitmap. Se si desidera un bitmap e il testo, si deve regolare il rettangolo in modo che sia abbastanza grande per entrambi.

Per modificare le dimensioni di un elemento owner-draw, occorre collegare un gestore all'evento a misura di elemento nel controllo owner-draw. Il nome dell'evento varia in funzione del controllo. Le caselle di riepilogo e quelle combinate usano *OnMeasureItem*. Le griglie non hanno alcun evento a misura di elemento.

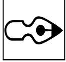
Il dimensionamento di un evento ha due importanti parametri: il numero di indice dell'elemento e le dimensioni dell'elemento stesso. Le dimensioni sono variabili: l'applicazione può crearle più piccole o più grandi. Le posizioni degli elementi successivi dipendono dalle dimensioni di quelli precedenti.

Per esempio, in una casella di riepilogo owner-draw variabile, se l'applicazione imposta l'altezza del primo elemento a cinque pixel, il secondo elemento disterà sei pixel, e così via. Sia nelle caselle di riepilogo sia in quelle combinate l'unico aspetto dell'elemento che l'applicazione può cambiare è l'altezza. La larghezza è sempre quella del controllo.

Le griglie owner-draw non possono modificare le dimensioni delle loro celle quando disegnano. Le dimensioni di ciascuna riga e di ciascuna colonna vengono impostate prima di disegnare dalle proprietà *ColWidths* e *RowHeights*.

Il seguente codice, collegato all'evento *OnMeasureItem* di una casella di riepilogo owner-draw, aumenta l'altezza di ciascun elemento dell'elenco per contenere la sua bitmap associata.

```
void __fastcall TForm1::ListBox1MeasureItem(TWinControl *Control, int Index,
  int &Height)      // note that Height is passed by reference
{
  int BitmapHeight = ((TBitmap *)ListBox1->Items->Objects[Index])->Height + 2;
  // make sure list item has enough room for bitmap (plus 2)
  if (BitmapHeight > Height)
    Height = BitmapHeight;
}
```



Occorre convertire gli elementi dalla proprietà *Objects* nell'elenco di stringhe. *Objects* è una proprietà di tipo *TObject* che pertanto può contenere qualsiasi tipo di oggetto. Quando si recuperano gli oggetti dall'array, occorre riconvertirli nel tipo effettivo degli elementi.

Disegno di elementi owner-draw

Quando un'applicazione ha bisogno di disegnare o di ridisegnare un controllo owner-draw, il sistema operativo genera eventi draw-item per ciascun elemento visibile del controllo. A seconda del controllo, l'elemento può anche ricevere gli eventi di tracciamento per l'elemento nel suo complesso o per i singoli sottoelementi.

Per disegnare un elemento di un controllo owner-draw, si deve collegare un gestore all'evento draw-item per quel controllo.

I nomi degli eventi per il disegno personalizzato di solito iniziano con uno dei seguenti modi:

- *OnDraw*, come *OnDrawItem* o *OnDrawCell*
- *OnCustomDraw*, come *OnCustomDrawItem*
- *OnAdvancedCustomDraw*, come *OnAdvancedCustomDrawItem*

L'evento draw-item contiene i parametri che indentificano l'elemento da disegnare, il rettangolo in cui disegnare e solitamente alcune informazioni sullo stato dell'elemento (per esempio, se l'elemento ha il fuoco). L'applicazione gestisce ciascun evento rappresentando l'elemento appropriato nel rettangolo dato.

Per esempio, il codice seguente mostra come disegnare gli elementi in una casella di riepilogo che ha delle bitmap associate a ciascuna stringa. Il codice collega questo gestore all'evento *OnDrawItem* per la casella di riepilogo:

```
void __fastcall TForm1::ListBox1DrawItem(TWinControl *Control, int Index,
  TRect &Rect, TOwnerDrawState State)

  TBitmap *Bitmap = (TBitmap *)ListBox1->Items->Objects[Index];
  ListBox1->Canvas->Draw(R.Left, R.Top + 2, Bitmap); // draw the bitmap
  ListBox1->Canvas->TextOut(R.Left + Bitmap->Width + 2, R.Top + 2,
    ListBox1->Items->Strings[Index]);        // and write the text to its right
}
```

Capitolo

7

Creazione di applicazioni, componenti e librerie

In questo capitolo viene fornita una panoramica sull'uso di C++Builder per la creazione di applicazioni, librerie e componenti.

Creazione di applicazioni

C++Builder è usato principalmente per la progettazione e la costruzione dei seguenti tipi di applicazioni:

- Applicazioni GUI
- Applicazioni per console
- Applicazioni di servizio (solo per applicazioni Windows)
- Package e DLL

Generalmente, le applicazioni GUI hanno un'interfaccia di facile impiego. Le applicazioni per console vengono eseguite da una finestra della console. Le applicazioni di servizio sono eseguite come servizi Windows. Questi tipi di applicazioni vengono compilati come eseguibili con codice di avvio.

Si possono creare altri tipi di progetto, come package e DLL, che portano alla realizzazione di package o di librerie collegabili dinamicamente. Queste applicazioni producono un codice eseguibile senza codice di avvio. Fare riferimento al paragrafo "Creazione di package e di DLL" on page 7-10.

Applicazioni GUI

Un'applicazione con interfaccia utente grafica (GUI) viene progettata utilizzando funzionalità grafiche come finestre, menu, finestre di dialogo e altre funzionalità che la rendono facile da usare. Quando si compila un'applicazione GUI, viene generato

un file eseguibile con codice di avvio. L'eseguibile generalmente fornisce la funzionalità di base del programma, tanto che i programmi più semplici sono costituiti esclusivamente da un file. È possibile estendere l'applicazione chiamando le DLL, i package e gli altri file di supporto dall'eseguibile.

C++Builder offre due modelli di applicazione UI:

- Single document interface (SDI)
- Multiple document interface (MDI)

Oltre che con il modello di implementazione delle applicazioni, il comportamento del progetto in fase di progettazione e di esecuzione dell'applicazione può essere modificato tramite le opzioni di progetto dell'IDE.

Modelli di interfaccia utente

Le schede possono essere implementate come schede MDI (Multiple document interface) o SDI (Single document interface). In un'applicazione MDI è possibile aprire all'interno di un'unica finestra genitore più di un documento o finestra figlio. Ciò vale per applicazioni come i fogli di lavoro o gli elaboratori testo. Un'applicazione SDI, invece, generalmente contiene un'unica vista del documento. Per rendere una scheda un'applicazione SDI, occorre impostare la proprietà *FormStyle* dell'oggetto *Form* a *fsNormal*.

Per ulteriori informazioni sullo sviluppo dell'interfaccia utente per un'applicazione, consultare il Capitolo 8, "Sviluppo dell'interfaccia utente dell'applicazione".

Applicazioni SDI

Per creare una nuova applicazione SDI:

1. Selezionare File | New | Other per richiamare la finestra di dialogo New Items.
2. Fare clic sulla pagina Projects e fare doppio clic su SDI Application.
3. Fare clic su OK.

Per impostazione predefinita, la proprietà *FormStyle* dell'oggetto *Form* è *fsNormal*, chioschi C++Builder presuppone che tutte le nuove applicazioni siano di tipo SDI.

Applicazioni MDI

Per creare una nuova applicazione MDI:

1. Selezionare File | New | Other per richiamare la finestra di dialogo New Items.
2. Fare clic sulla pagina Projects e fare doppio clic su MDI Application.
3. Fare clic su OK.

Le applicazioni MDI richiedono una maggiore pianificazione e sono in un certo senso più complesse da progettare di quelle SDI. Le applicazioni MDI generano finestre figlio che risiedono all'interno della finestra client; la scheda principale contiene le schede figlio. Per specificare se una scheda è figlio (*fsMDIForm*) o la scheda principale (*fsMDIChild*), occorre impostare la proprietà *FormStyle* dell'oggetto *TForm*. È consigliabile definire una classe base per le schede figlio e derivare ogni scheda figlio da questa classe, per evitare di dover ridefinire le proprietà della scheda figlio.

Le applicazioni MDI spesso includono sul menu principale una finestra popup che contiene elementi quali Cascade e Tile per la visualizzazione di più finestre in modalità differenti. Quando una finestra figlio viene ridotta a icona, la sua icona è situata nella scheda MDI genitore.

Per riassumere ciò che è necessario fare per creare le finestre per l'applicazione MDI:

1 Creare la scheda della finestra principale o finestra MDI genitore. Impostare la sua proprietà *FormStyle* a *fsMDIForm*.
2 Creare un menu per la finestra principale che include File|Open, File|Save e Window che abbia gli elementi Cascade, Tile e Arrange All.
3 Creare le schede MDI figlio e impostare le loro proprietà *FormStyle* a *fsMDIChild*.

Impostazioni delle opzioni per l'IDE, il progetto e la compilazione

Per specificare le diverse opzioni per il progetto, utilizzare il comando Project| Options. Per maggiori informazioni, consultare la Guida in linea.

Impostazione delle opzioni predefinite del progetto

Per cambiare le opzioni predefinite che valgono per tutti i progetti, si deve ricorrere alla finestra di dialogo Project Options e contrassegnare la casella Default nell'angolo in basso a destra. Per impostazione predefinita, tutti i nuovi progetti utilizzeranno le opzioni correnti selezionate.

Modelli di programmazione

I modelli di programmazione sono *scheletri* di strutture comunemente utilizzate, che è possibile aggiungere al codice sorgente e quindi completare. In C++Builder sono inclusi alcuni modelli di codice standard, come quelli per gli array, per le classi, per la dichiarazioni di funzioni, nonché molte istruzioni.

È possibile anche scrivere dei modelli propri per codificare delle strutture di impiego frequente. Ad esempio, se si desidera utilizzare nel codice un ciclo **for**, si può inserire il seguente modello:

```
for (; ;)
{

}
```

Per inserire un modello di codice nel Code editor, premere *Ctrl-j* e selezionare il modello di codice che si intende utilizzare. A questa raccolta è possibile aggiungere anche dei propri modelli. Per aggiungere un modello:

1 Scegliere il comando Tools|Editor Options.
2 Fare clic sul separatore Code Insight.
3 Nei sezione Template, fare clic su Add.
4 Dopo Shortcut name scrivere un nome per il modello, immettere una breve descrizione del nuovo modello e fare clic su OK.

5 Aggiungere il codice del modello nella casella di testo Code.
6 Fare clic su OK.

Applicazioni per console

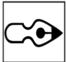
Le applicazioni per console sono programmi a 32-bit che vengono eseguiti senza un'interfaccia grafica, solitamente in una finestra per console. Queste applicazioni di solito non richiedono un'interazione con l'utente ed eseguono un numero limitato di funzioni.

Per creare una nuova applicazione per console:

1 Selezionare l'opzione di menu File | New | Other ,quindi nella finestra di dialogo New Items selezionare l'opzione Console Wizard.
2 Nella finestra di dialogo Console Wizard, selezionare l'opzione Console Application, scegliere il tipo di file sorgente (C o C++) per il modulo principale del progetto, o specificare un file esistente che contenga una funzione main o winmain, quindi fare clic sul pulsante OK.

C++Builder creerà un file di progetto per questo tipo di file sorgente e visualizzerà il Code editor.

Utilizzo di VCL e CLX in applicazioni per console



Quando si crea una nuova applicazione per console, l'IDE non crea una nuova scheda. Viene visualizzato solo il Code editor.

Nelle applicazioni per console è possibile tuttavia utilizzare la VCL e la CLX. Per fare ciò occorre indicare nel Console Wizard che si intende utilizzare la VCL o la CLX (selezionare l'opzione Use VCL oppure Use CLX). Se nel Wizard non si indica che si vuole utilizzare la VCL o la CLX, in seguito non sarà più possibile utilizzare alcuna classe della VCL o della CLX in quella applicazione. Se si tenta di farlo, il linker genererà una serie di errori.

Le applicazioni per console dovrebbero gestire tutte le eccezioni per impedire alle finestre di visualizzare una finestra di dialogo durante la sua esecuzione.

Applicazioni di servizio

Le applicazioni di servizio accettano richieste dalle applicazioni client, elaborano tali richieste e restituiscono informazioni all'applicazione client. Solitamente vengono eseguite in background, con poca o nulla interazione con l'utente. Esempi di applicazioni di servizio sono i server Web, i server FTP o di posta.

Per creare un'applicazione che implementi un servizio Win32:

1 Scegliere File | New | Other e fare doppio clic su Service *Application* nella finestra di dialogo New Items. In questo modo si aggiunge una variabile di nome *Application* al progetto, che è di tipo *TServiceApplication*.

2 Apparirà una finestra Service che corrisponde a un servizio (*TService*). Per implementare il servizio, occorre impostarne le proprietà e i gestori di evento nell'Object Inspector.

3 Scegliendo File|New|Other e selezionando Service dalla finestra di dialogo New Items, è possibile aggiungere ulteriori servizi all'applicazione del server. Non è possibile aggiungere servizi ad un'applicazione che non sia di servizio. Anche se è possibile aggiungere un oggetto *TService*, l'applicazione non genererà gli eventi necessari, né effettuerà le chiamate appropriate ad Windows a favore del servizio.

4 Una volta creata l' applicazione di servizio, è possibile installarne i servizi con Service Control Manager (SCM). Le altre applicazioni potranno quindi attivare tali servizi inviando delle richieste a SCM.

Per installare i servizi dell'applicazione, eseguire il programma utilizzando l'opzione /INSTALL. L'applicazione installerà i suoi servizi e terminerà, emettendo un messaggio di conferma dell'avvenuta installazione dei servizi. Se si esegue l'applicazioni di servizio utilizzando l'opzione /SILENT è possibile evitare la visualizzazione del messaggio di conferma.

Per disinstallare i servizi, eseguire l'applicazione dalla riga comandi utilizzando l'opzione /UNINSTALL. (È possibile utilizzare anche l'opzione /SILENT per sopprimere il messaggio di conferma emesso in fase di disinstallazione).

Esempio Questo servizio ha un *TServerSocket* la cui porta è impostata a 80. Questa è la porta predefinita dei browser Web per effettuare richieste ai server Web e dei server Web per fornire risposte ai browser. Questo particolare esempio produce un documento di testo di nome WebLogxxx.log (dove *xxx* è ThreadID) nella directory C:\Temp. Ci deve essere un unico server in ascolto nella porta stabilita, in modo che, se c'è un server Web, si avrà la certezza che non sta ascoltando (il servizio viene interrotto).

Per vedere il risultato: aprire un browser Web sulla macchina locale e immettere 'localhost' (senza virgolette) per l'indirizzo. Probabilmente il browser si disattiverà, ma a questo punto sarà disponibile un file di nome Weblogxxx.log nella directory C:\Temp.

1 Per creare l'esempio, scegliere il comando File|New |Other e selezionare nella finestra di dialogo New Items l'opzione Service *Application.* Appare la finestra Service1.

2 Aggiungere alla finestra del servizio (Service1) un componente ServerSocket prelevandolo dalla pagina Internet della Component palette).

3 Aggiungere alla classe TService1 un membro dati privato di tipo TMemoryStream. L' intestazione della unit dovrebbe assomigliare al seguente:

```
//---------------------------------------------------------------------------
#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <SysUtils.hpp>
#include <Classes.hpp>
#include <SvcMgr.hpp>
#include <ScktComp.hpp>
//---------------------------------------------------------------------------
```

```
class TService1 : public TService
{
__published:// IDE-managed Components
        TServerSocket *ServerSocket1;
private:// User declarations
        TMemoryStream *Stream; // add this line here
public:// User declarations
   __fastcall TService1(TComponent* Owner);
   PServiceController __fastcall GetServiceController(void);

   friend void __stdcall ServiceController(unsigned CtrlCode);
};
//---------------------------------------------------------------------------
extern PACKAGE TService1 *Service1;
//---------------------------------------------------------------------------
#endif
```

4 Selezionare ServerSocket *1,* il componente aggiunto al passo *1.* Fare doppio clic sull'evento *OnClientRead* nell'Object Inspector ed aggiungere il seguente gestore di evento:

```
void __fastcall TService1::ServerSocket1ClientRead(TObject *Sender,
      TCustomWinSocket *Socket)
{
  char *Buffer = NULL;
  int len = Socket->ReceiveLength();
  while (len > 0)
  {
    try
    {
      Buffer = (char *)malloc(len);
      Socket->ReceiveBuf((void *)Buffer, len);
      Stream->Write(Buffer, len);
    }
    __finally
    {
      free(Buffer);
    }
    Stream->Seek(0, soFromBeginning);
    AnsiString LogFile = "C:\\Temp\\WebLog";
    LogFile = LogFile + IntToStr(ServiceThread->ThreadID) + ".log";
    Stream->SaveToFile(LogFile);
  }
}
```

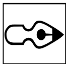
5 Infine, selezionare Service1 facendo clic nell'area del client della finestra (non sul ServiceSocket). Fare doppio clic sull'evento *OnExecute* nell'Object Inspector ed aggiungere il seguente gestore di evento:

```
void __fastcall TService1::Service1Execute(TService *Sender)
{
  Stream = new TMemoryStream();
  try
  {
    ServerSocket1->Port = 80; // WWW port
```

```
    ServerSocket1->Active = true;
    while (!Terminated)
      ServiceThread->ProcessRequests(true);
    ServerSocket1->Active = false;
  }
  __finally
  {
    delete Stream;
  }
}
```

Quando si scrivono le applicazioni di servizio, bisogna far attenzione a:

- Thread dei servizi
- Proprietà del nome del servizio
- Debug delle applicazioni di servizio



Le applicazioni di servizio sono solo per l'ambiente Windows.

Thread dei servizi

Ogni servizio ha un proprio thread (*TServiceThread*), cosicché se l'applicazione implementa più di un servizio, occorre verificare che l'implementazione dei servizi sia a prova di thread. *TServiceThread* è progettato in modo che sia possibile implementare il servizio nel gestore di evento *TService OnExecute*. Il thread del servizio ha un proprio metodo *Execute* che contiene un ciclo che chiama i gestori *OnStart* e *OnExecute* del servizio prima di prendere in considerazione nuove richieste.

Dal momento che i servizi richiesti possono richiedere molto tempo per l'elaborazione e l'applicazione che fornisce il servizio può ricevere richieste simultanee da più di un client, è più efficiente derivare un nuovo thread (da *TThread*, non da *TServiceThread*) per ciascuna richiesta e spostare l'implementazione di quel servizio al metodo *Execute* del nuovo thread. Ciò consente al ciclo *Execute* del thread di servizio di elaborare in continuazione le nuove richieste, senza dover attendere che il gestore *OnExecute* del servizio termini. L'esempio seguente illustra quanto detto finora.

Esempio

Questo servizio emette un segnale acustico ogni 500 millisecondi dall'interno del thread standard. Esso gestisce la sospensione, la prosecuzione e l'interruzione del thread quando al sevizio viene chiesto di sospenderlo, continuarlo o interromperlo.

1. Scegliere File | New | Other e selezionare dalla finestra di dialogo New Items l'opzione Service Application. Apparirà la finestra Service1.
2. Nel file header della unit, dichiarare un nuovo discendente di *TThread* di nome TSparkyThread. Questo è il thread che esegue i compiti affidati al servizio. La dichiarazione dovrebbe essere come segue:

```
class TSparkyThread : public TThread
{
private:
protected:
        void __fastcall Execute();
public:
```

```
        __fastcall TSparkyThread(bool CreateSuspended);
};
```

3 Nel file .cpp della unit, creare una variabile globale per un'istanza di TSparkyThread:

```
TSparkyThread *SparkyThread;
```

4 Aggiungere il codice seguente al file .cpp del costruttore TSparkyThread:

```
__fastcall TSparkyThread::TSparkyThread(bool CreateSuspended)
        : TThread(CreateSuspended)
{
}
```

5 Aggiungere il codice seguente al file .cpp del metodo *Execute* di TSparkyThread (la funzione di thread):

```
void __fastcall TSparkyThread::Execute()
{
  while (!Terminated)
  {
    Beep();
    Sleep(500);
  }
}
```

6 Selezionare la finestra del servizio (Service1), e fare doppio clic sull'evento *OnStart* nell'Object Inspector. Aggiungere il seguente gestore di evento OnStart:

```
void __fastcall TService1::Service1Start(TService *Sender, bool &Started)
{
  SparkyThread = new TSparkyThread(false);
  Started = true;
}
```

7 Fare doppio clic sull'evento *OnContinue* nell'Object Inspector. Aggiungere il seguente gestore di evento OnContinue:

```
void __fastcall TService1::Service1Continue(TService *Sender, bool &Continued)
{
  SparkyThread->Resume();
  Continued = true;
}
```

8 Fare doppio clic sull'evento *OnPause* nell'Object Inspector. Aggiungere il seguente gestore di evento OnPause:

```
void __fastcall TService1::Service1Pause(TService *Sender, bool &Paused)
{
  SparkyThread->Suspend();
  Paused = true;
}
```

9 Infine, fare doppio clic sull'evento *OnStop* nell'Object Inspector e aggiungere il seguente gestore di evento OnStop:

```
void __fastcall TService1::Service1Stop(TService *Sender, bool &Stopped)
{
```

```
    SparkyThread->Terminate();
    Stopped = true;
  }
```

Quando si sviluppano applicazioni server, la scelta di generare un nuovo thread dipende dalla natura del servizio da fornire, dal numero anticipato di connessioni e dal numero previsto di processori nel computer che esegue il servizio.

Proprietà del nome del servizio

La VCL fornisce le classi per la creazione di applicazioni di servizio per la piattaforma Windows (non disponibile per applicazioni multipiattaforma). Ne fanno parte *TService* e *TDependency.* Quando si utilizzano queste classi, è possibile fare confusione tra le diverse proprietà del nome. In questa sezione vengono descritte le differenze.

I servizi hanno nomi utente (denominati Service start names) associati a password, nomi per la visualizzazione nelle finestre manager e editor, e nomi veri e propri (il nome del servizio). Le dipendenze possono essere servizi o possono essere caricate ordinando i gruppi. Esse anche nomi e nomi di visualizzazione. E, dal momento che gli oggetti service sono derivati da *TComponent*, essi derivano la proprietà *Name.* Le sezioni successive riportano le proprietà dei nomi:

Proprietà di TDependency

DisplayName di *TDependency* è sia un nome di visualizzazione sia il nome vero e proprio del servizio. Coincide quasi sempre con la proprietà *Name* di *TDependency*.

Proprietà Name di TService

La proprietà *Name* di *TService* viene ereditata da *TComponent*.È il nome del componente ed anche il nome del servizio. Per le dipendenze che sono servizi, questa proprietà coincide con le proprietà *Name* e *DisplayName* di *TDependency*.

DisplayName di *TService* è il nome visualizzato nella finestra Service Manager. Spesso è diverso dal nome effettivo del servizio (*TService::Name*, *TDependency::DisplayName*, *TDependency::Name*). Si noti che il *DisplayName* per la dipendenza e il *DisplayName* per il servizio generalmente sono diversi.

I Service start names si distinguono sia dai nomi di visualizzazione del servizio sia dai nomi effettivi del servizio. Un *ServiceStartName* è il nome utente inserito nella finestra di dialogo Start selezionata dal Service Control Manager.

Debug delle applicazioni di servizio

È possibile eseguire il debug di applicazioni di servizio collegandosi al processo dell'applicazione di servizio quando è già in esecuzione (cioè, avviando per prima cosa il servizio e quindi collegandosi al debugger). Per agganciarsi al processo dell'applicazione di servizio, selezionare il comando Run | Attach To Process e selezionare nella finestra di dialogo che verrà attivata l'applicazione di servizio.

In alcuni casi, questo secondo metodo potrebbe non riuscire poiché si hanno diritti insufficienti. In questo caso, si può utilizzare Service Control Manager per abilitare il servizio a a funzionare con il debugger:

1 Per prima cosa, nella posizione del file di registro di Windows indicata di seguito, creare una chiave di nome **Image File Execution Options**:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion
```

2 Creare una sottochiave avente lo stesso nome del servizio (ad esempio, MYSERV.EXE). A questa sottochiave aggiungere un valore di nome Debugger e di tipo REG_SZ. Come valore della stringa, utilizzare il nome di percorso completo del BCB.exe.

3 Nel programma Services control panel, selezionare il servizio, fare clic su Startup e selezionare l'opzione Allow Service to Interact with Desktop.

Sui sistemi Windows NT, è possibile utilizzare un altro metodo per eseguire il debug di applicazioni di servizio. Tuttavia, questo approccio può risultare complicato in quanto richiede tempi di esecuzione molto brevi:

1 Per prima cosa, avviare l'applicazione nel debugger. Attendere qualche secondo affinché l'applicazione sia caricata completamente.

2 Avviare rapidamente il servizio dal pannello di controllo o dalla riga comandi:

```
start MyServ
```

Il servizio deve essere avviato rapidamente (entro 15-30 secondi dall'avvio dell'applicazione) altrimenti, se non viene avviato alcun servizio, l'applicazione terminerà.

Creazione di package e di DLL

Le librerie a collegamento dinamico (DLL) sono moduli di codice compilato che funziona in abbinamento con un eseguibile per fornire funzionalità ad una applicazione. È possibile creare delle DLL nei programmi multipiattaforma.Tuttavia, in ambiente Linux, le DLL (e i package) vengono ricompilati come oggetti condivisi.

I package sono speciali DLL usate dalle applicazioni C++Builder, dall'IDE o da entrambe. Esistono due tipi di package: i package di esecuzione e quelli di progettazione. I package di esecuzione forniscono funzionalità ad un programma durante il sua funzionamento. I package di progettazione estendono la funzionalità dell'IDE.

Le DLL e le librerie dovrebbero gestire tutte le eccezioni per evitare la visualizzazione di errori e di avvertimenti mediante le finestre di dialogo di Windows.

Nel file di progetto della libreria è possibile introdurre le seguenti direttive al compilatore:

Tabella 7.1 Direttive al compilatore per le librerie

Direttiva al compilatore	Descrizione
{$LIBPREFIX 'string'}	Aggiunge uno specifico prefisso al nome del file di output. Ad esempio, si potrebbe specificare {$LIBPREFIX 'dcl'} per un package di progettazione oppure utilizzare {$LIBPREFIX ' ' } per eliminare completamente il prefisso.
{$LIBSUFFIX 'string'}	Aggiunge uno specifico suffisso al nome del file di output prima dell'estensione. Ad esempio, utilizzare {$SOSUFFIX '-2.1.3' } in something.cpp per generare something-2.1.3.bpl.
{$LIBVERSION 'string'}	Aggiunge una seconda estensione al nome del file di output dopo l'estensione .bpl. Ad esempio, utilizzare {$SOSUFFIX '-2.1.3' } in something.cpp per generare something-2.1.3.bpl.

Per ulteriori informazioni sui package, consultare il Capitolo 15, "Uso di package e di componenti".

Quando usare i package e le DLL

Per la maggior parte delle applicazioni scritte in C++Builder, i package forniscono una maggiore flessibilità e sono più facili da creare delle DLL. Tuttavia, ci sono diverse situazioni in cui le DLL risultano più adatte dei package per i progetti:

- Quando il modulo di codice verrà chiamato da applicazioni non C++Builder.
- Se si estende la funzionalità di un server Web.
- Quando si crea un modulo di codice che deve essere utilizzato da sviluppatori terzi.
- Quando il progetto è un contenitore OLE.

Non si possono passare le informazioni di tipo in esecuzione (RTTI) da una DLL a un'altra, oppure da una DLL a un eseguibile. Questo perché le DLL gestiscono tutte le proprie informazioni sui simboli. Se è necessario passare un oggetto *TStrings* da una DLL, utilizzando un operatore **is** oppure **as**, bisogna creare un package invece di una DLL. I package condividono le informazioni sui simboli.

Uso delle DLL in C++Builder

Le DLL di Windows possono essere utilizzate nelle applicazioni C++Builder proprio come nelle normali applicazioni C++.

Per caricare in modo statico una DLL al caricamento di un'applicazione C++Builder, occorre collegare all'applicazione C++Builder il file di libreria di importazione per la DLL durante il linking. Per aggiungere una libreria di importazione a un'applicazione C++Builder, scegliere Project | Add to Project e selezionare il file .LIB che si desidera aggiungere.

Le funzioni esportate della DLL vengono quindi rese disponibili all'applicazione. Mediante il modificatore **__declspec** (dllimport)creare un prototipo delle funzioni DLL utilizzate dall'applicazione:

```
__declspec(dllimport) return_type imported_function_name(parameters);
```

Per caricare in modo dinamico una DLL durante l'esecuzione di un'applicazione C++Builder, includere la libreria di importazione, esattamente come nel caso del caricamento statico, e impostare l'opzione del linker per il caricamento ritardato nella pagina Project|Options|Advanced Linker. Si può anche ricorrere alla funzione API di Windows *LoadLibrary()*, che carica la DLL, e poi alla funzione API *GetProcAddress()*, che ricava i puntatori alle singole funzioni che si vogliono utilizzare.

Ulteriori informazioni sull'impiego delle DLL si trovano in *Microsoft® Win32 SDK Reference*.

Creazione di DLL in C++Builder

La creazione di DLL in C++Builder è analoga alla creazione in C++ standard:

1. Scegliere il comando File|New|Other per visualizzare la finestra di dialogo New Items.
2. Fare doppio clic sull'icona DLL Wizard.
3. Scegliere il tipo di sorgente per il modulo principale (C o C++).
4. Se si vuole che l'entry point della DLL sia DllMain, secondo lo stile MSVC++, selezionare l'opzione VC++ style, altrimenti, come entry point sarà utilizzato DLLEntryPoint.
5. Fare clic su Use VCL o su Use CLX per creare una DLL contenente componenti della VCL o della CLX. L'opzione è disponibile solo per moduli sorgente in C++. Consultare "Creazione di DLL contenenti componenti VCL e CLX" on page 7-13.
6. Se si vuole che la DLL sia multi-thread, selezionare l'opzione Multi-threaded.
7. Fare clic su OK.

Le funzioni esportate presenti nel codice devono essere identificate mediante il modificatore **__declspec** (dllexport) proprio come avviene in Borland C++ o in Microsoft Visual C++. Per esempio, il codice seguente è consentito sia in C++Builder sia in altri compilatori Windows C++:

```
// MyDLL.cpp
double dblValue(double);
double halfValue(double);
extern "C" __declspec(dllexport) double changeValue(double, bool);

double dblValue(double value)
{
   return value * value;
};

double halfValue(double value)
```

```
{
   return value / 2.0;
}

double changeValue(double value, bool whichOp)
{
   return whichOp ? dblValue(value) : halfValue(value);
}
```

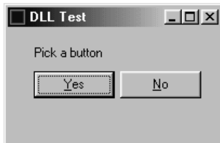
Nel codice precedente viene esportata la funzione *changeValue*, che viene resa pertanto disponibile alle applicazioni che la chiamano. Le funzioni *dblValue* e *halfValue* sono interne, per cui non possono essere chiamate dall'esterno della DLL.

Ulteriori informazioni sulla creazione di DLL si trovano in *Microsoft® Win32 SDK Reference*.

Creazione di DLL contenenti componenti VCL e CLX

Uno dei punti di forza delle DLL è il fatto che una DLL creata con un determinato strumento di sviluppo può spesso essere utilizzata da un'applicazione scritta con uno strumento di sviluppo differente. Quando la DLL contiene dei componenti VCL (per esempio, delle schede) che devono essere utilizzate dall'applicazione chiamante, bisogna preparare delle routine di interfaccia esportate che utilizzino le convenzioni di chiamata standard, evitare il mangling dei nomi di C++, mentre non è richiesto che l'applicazione chiamante supporti la libreria VCL o CLX per poter funzionare. Per creare componenti VCL o CLX che possano essere esportati, utilizzare i package di runtime. Per ulteriori informazioni, consultare il Capitolo 15, "Uso di package e di componenti".

Si supponga, per esempio, di voler creare una DLL per visualizzare questa semplice finestra di dialogo:



Il codice della DLL della finestra di dialogo è il seguente:

```
// DLLMAIN.H
//---------------------------------------------------------------------
#ifndef dllMainH
#define dllMainH
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
//---------------------------------------------------------------------
class TYesNoDialog : public TForm
{
```

```
__published:    // IDE-managed Components
   TLabel *LabelText;
   TButton *YesButton;
   TButton *NoButton;
   void __fastcall YesButtonClick(TObject *Sender);
   void __fastcall NoButtonClick(TObject *Sender);
private:        // User declarations
   bool returnValue;
public:         // User declarations
   virtual __fastcall TYesNoDialog(TComponent *Owner);
   bool __fastcall GetReturnValue();
};

// exported interface function
extern "C" __declspec(dllexport) bool InvokeYesNoDialog();

//----------------------------------------------------------------------
 extern TYesNoDialog *YesNoDialog;
//----------------------------------------------------------------------
#endif


// DLLMAIN.CPP
//----------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "dllMain.h"
//----------------------------------------------------------------------
#pragma resource "*.dfm"
TYesNoDialog *YesNoDialog;
//----------------------------------------------------------------------
__fastcall TYesNoDialog::TYesNoDialog(TComponent *Owner)
 : TForm(Owner)
{
  returnValue = false;
}
//----------------------------------------------------------------------
void __fastcall TYesNoDialog::YesButtonClick(TObject *Sender)
{
   returnValue = true;
   Close();
}
//----------------------------------------------------------------------
void __fastcall TYesNoDialog::NoButtonClick(TObject *Sender)
{
   returnValue = false;
   Close();
}
//----------------------------------------------------------------------
bool __fastcall TYesNoDialog::GetReturnValue()
{
   return returnValue;
}
//----------------------------------------------------------------------
```

```
// exported standard C++ interface function that calls into VCL
bool InvokeYesNoDialog()
{
   bool returnValue;
      TYesNoDialog *YesNoDialog = new TYesNoDialog(NULL);
   YesNoDialog->ShowModal();
      returnValue = YesNoDialog->GetReturnValue();
      delete YesNoDialog;
   return returnValue;
}
//----------------------------------------------------------------------
```

Il codice riportato nell'esempio visualizza la finestra di dialogo e, se viene premuto il pulsante "Yes", memorizza il valore **true** nel membro dati privato *returnValue*. In caso contrario, *returnValue* ha valore **false**. La funzione pubblica *GetReturnValue()* recupera il valore corrente di *returnValue*.

Per richiamare la finestra di dialogo e stabilire quale pulsante è stato premuto, l'applicazione chiamante chiama la funzione esportata *InvokeYesNoDialog()*. Questa funzione è dichiarata in DLLMAIN.H come funzione esportata utilizzando le convenzioni standard del C per il collegamento (in modo da evitare il mangling dei nomi di C++) e per la chiamata. La funzione è definita in DLLMAIN.CPP.

Grazie all'utilizzo di una funzione C standard come interfaccia verso la DLL, qualsiasi applicazione chiamante, creata o meno in C++Builder, può utilizzare la DLL. Le funzionalità VCL e CLX richieste per supportare la finestra di dialogo è collegata nella DLL stessa, e l'applicazione chiamante non ha bisogno di ulteriori informazioni a tale proposito.

Si tenga presente che, quando si crea una DLL che utilizza la VCL o la CLX, i componenti necessari della VCL o della CLX vengono collegati alla DLL, dando luogo ad un certo appesantimento. L'impatto di tale appesantimento sulle dimensioni totali dell'applicazione può essere ridotto al minimo combinando più componenti in un'unica DLL, la quale necessita di una sola copia dei componenti di supporto della VCL o della CLX.

Collegamento delle DLL

È possibile impostare le opzioni del linker per le DLL utilizzando la pagina Linker della finestra di dialogo Project Options. La casella di controllo predefinita su questa pagina crea inoltre una libreria di importazione per la DLL. Se si effettua la compilazione dalla riga comandi, chiamare il linker, ILINK32.EXE, utilizzando l'opzione `-Tpd`. Ad esempio:

```
ilink32 /c /aa /Tpd c0d32.obj mydll.obj, mydll.dll, mydll.map, import32.lib cw32mt.lib
```

Se occorre generare una libreria di importazione, utilizzare anche l'opzione -Gi.

Se occorre, è possibile creare una libreria di importazione utilizzando il programma di utilità dalla riga comandi IMPLIB.EXE. Ad esempio:

```
implib mydll.lib mydll.dll
```

Per maggiori informazioni sulle diverse opzioni per il collegamento delle DLL e sull'utilizzo delle stesse con altri moduli collegati in modo statico o dinamico alla libreria di runtime, consultare la Guida in linea.

Scrittura di applicazioni database

Uno dei punti di forza di C++Builder è il suo supporto per la creazione di applicazioni database evolute. C++Builder possiede una serie di strumenti interni che consentono di collegarsi a server SQL e a database Oracle, Sybase, InterBase, MySQL, MS-SQL, Informix e DB2, fornendo una condivisione trasparente dei dati tra le varie applicazioni.

C++Builder include molti componenti per accedere ai database e rappresentare le informazioni in essi contenute. Sulla Component palette, i componenti database sono raggruppati in base alla funzione svolta e secondo il meccanismo di accesso di dati.

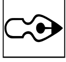
Tabella 7.2 Pagine di database sulla Component palette

Pagina della Component palette	Contenuti
BDE	Componenti che utilizzano Borland Database Engine (BDE), una vasta API per interagire con i database. BDE supporta una vasta gamma di funzioni ed è fornito con molti programmi di utilità tra i quali Database Desktop, Database Explorer, SQL Monitor e BDE Administrator. Per i particolari, vedere il Capitolo 24, "Uso di Borland Database Engine".
ADO	Componenti che utilizzano ActiveX Data Objects (ADO), sviluppato da *Microsoft*, per accedere alle informazioni di un database. Sono disponibili molti driver ADO per la connessione a vari server di database. I componenti basati su ADO permettono di integrare l'applicazione in un ambiente ADO. Per i particolari, vedere il Capitolo 25, "Operazioni con componenti ADO".
dbExpress	Componenti multipiattaforma che utilizzano dbExpress per accedere alle informazioni nel database. I driver dbExpress consentono un accesso rapido ai database ma, per eseguire gli aggiornamenti, devono essere utilizzati con *TClientDataSet* e *TDataSetProvider*. Per i dettagli vedere il Capitolo 26, "Uso di dataset unidirezionali".
InterBase	Componenti per l'accesso diretto ai database InterBase, senza passare attraverso uno strato separato di motore. Per ulteriori informazioni sull'utilizzo dei componenti InterBase, consultare la Guida in linea.
Data Access	Componenti, come *TClientDataSet* e *TDataSetProvider*, che possono essere utilizzati con qualsiasi meccanismo di accesso ai dati. Per informazioni sui dataset client, consultare il Capitolo 27, "Utilizzo dei dataset client". Per informazioni sui provider, consultare il Capitolo 28, "Uso di componenti provider".
Data Controls	Controlli data-aware che possono accedere alle informazioni provenienti da un datasource. Consultare il Capitolo 19, "Uso dei controlli dati".

Quando si progetta un'applicazione database, è necessario decidere che meccanismo di accesso ai dati utilizzare. Ogni meccanismo di accesso ai dati differisce dagli altri

per ampiezza di supporto funzionale, per semplicità di distribuzione e nella disponibilità di driver per il supporto dei diversi server di database.

Per informazioni dettagliate sull'uso di C++Builder per creare sia applicazioni client database sia server di applicazioni, consultare la Parte II, "Sviluppo di applicazioni database" in questo stesso manuale. Per informazioni sulla ditribuzione, fare riferimento a "Distribuzione di applicazioni database" a pagina 17-6.



Non tutte le edizioni di C++Builder includono il supporto per i database.

Distribuzione di applicazioni database

C++Builder fornisce il supporto per la creazione di applicazioni database distribuite grazie all'utilizzo di un set coordinato di componenti. Le applicazioni database distribuite possono essere costruite su un gran numero di protocolli di comunicazione, compresi DCOM, TCP/IP e SOAP.

Per ulteriori informazioni sulla costruzione di applicazioni database distribuite, consultare il Capitolo 29, "Creazione di applicazioni multi-tier".

La distribuzione di applicazioni database spesso richiede la fornitura di Borland Database Engine (BDE) in aggiunta ai file dell'applicazione. Per informazioni sulla distribuzione del BDE, consultare "Distribuzione di applicazioni database" a pagina 17-6.

Creazione di applicazioni per Web server

Le applicazioni per Web server sono applicazioni che vengono eseguite su server per la distribuizione su Internet di contenuti Web come pagine Web HTML o documenti XML. Tra gli esempi di applicazioni per Web server vi sono quelle per controllare l'accesso a un sito Web, per generare ordini di acquisto o per rispondere a richieste di informazioni.

Sfruttando le seguenti tecnologie C++Bilder si possono creare vari tipi di applicazioni per Web server:

- Web Broker
- WebSnap
- InternetExpress
- Web Services

Utilizzo di Web Broker

Si utilizza Web Broker (detto anche architettura NetCLX) per creare applicazioni per Web server, come applicazioni CGI o librerie a collegamento dinamico (DLL). Queste applicazioni per Web server possono contenere qualsiasi componente di tipo non visuale. I componenti raccolti nella pagina Internet della Component palette permettono di creare gestori di evento, di costruire da programma documenti HTML o XML e di trasferirli al client.

Per creare una nuova applicazione per server Web utilizzando l'architettura Web Broker, selezionare il comando File | New | Other, quindi Web Server *Application* nella finestra di dialogo New Items. Infine, si deve scegliere il tipo di applicazione per server Web:

Tabella 7.3 Applicazioni per Web server

Tipo di applicazione Web server	Descrizione
ISAPI and NSAPI Dynamic Link Library	Le applicazioni per server Web ISAPI o NSAPI sono DLL caricate dal server Web. Le informazioni di richiesta del client vengono passate alla DLL come struttura e valutate da TISAPIApplication. Ogni messaggio di richiesta è gestito in un thread di esecuzione separato. La selezione di questo tipo di applicazione provoca l'aggiunta dell'heder di libreria dei file di progetto e di tutte le voci necessarie all'elenco uses e alla clausola export del file di progetto.
CGI Stand-alone executable	Le applicazioni per server Web CGI sono applicazione per console che ricevono richieste dal client sull'input standard, le elaborano e restituiscono il risultato al server sull'output standard affinché le invii al client.
Win-CGI Stand-alone executable	Le applicazioni per server Web Win-CGI sono applicazioni Windows che ricevono richieste dai client mediante un file di impostazioni di configuazione (INI) scritto dal server, e scrivono il risultato in un file che il server restituisce al client. Il file INI è valutato da TCGIApplication. Ogni messaggio di richiesta è gestito da un'istanza separata dell'applicazione.
Apache Shared Module (DLL)	La selezione di questo tipo di applicazione imposta il progetto come una DLL. Le applicazioni per server Web Apache sono DLL caricate dal server Web. Le informazioni vengono passate alla DLL, elaborate e restituite al client dal server Web.
Web App Debugger Stand-alone executable	La selezione di questo tipo di applicazione imposta l'ambiente per lo sviluppo e il collaudo di applicazioni per server Web. Le applicazioni Web App Debugger sono file eseguibili caricati dal server Web. Questo tipo di applicazione non è destinato alla distribuzione.

Le applicazioni CGI e Win-CGI usano un maggior numero di risorse di sistema sul server, pertanto è preferibile che le applicazioni complesse vengano create come applicazioni ISAPI, NSAPI o DLL Apache. Se si scrivono applicazioni multipiattaforma, per lo sviluppo per Web server si deve selezionare l'opzione CGI stand-alone oppure Apache Shared Module (DLL). Sono le stesse opzioni che compaiono quando si creano applicazioni WebSnap e Web Service.

Per ulteriori informazioni sulla costruzione di applicazioni per server Web, consultare il Capitolo 32, "Creazione di applicazioni server per Internet".

Creazione di applicazioni WebSnap

WebSnap fornisce un insieme di componenti e di wizard per la realizzazione di Web server evoluti che interagiscono con i Web browsers. I componenti WebSnap generano contenuti HTML o altri contenuti MIME per pagine Web. WebSnap è per lo

sviluppo lato server. Attualmente, WebSnap non può essere utilizzato in applicazioni multipiattaforma.

Per creare una nuova applicazione WebSnap, selezionare il comando File | New | Other, quindi selezionare la pagina WebSnap nella finestra di dialogo New Items. Scegliere WebSnap *Application*. Infine, si deve scegliere il tipo di applicazione per server Web: (ISAPI/NSAPI, CGI, Win-CGI, Apache). Per i dettagli, consultare la Tabella 7.3, "Applicazioni per Web server".

Per ulteriori informazioni su WebSnap, consultare il Capitolo 34, "Creazione di applicazioni Web Server con WebSnap".

Utilizzo di InternetExpress

InternetExpress è un set di componenti che arricchisce l'architettura di base per applicazioni per Web server, consentendole di agire come client di un application server. Si utilizza InternetExpress per applicazioni in cui i client basati su browser prelevano dati da un provider, risolvono gli aggiornamenti sul provider, pur essendo in esecuzione su un client.

Le applicazioni InternetExpress generano pagine HTML che contengono un misto di HTML, XML e javascript. HTML determina la disposizione e l'aspetto delle pagine visualizzate nel browser dell'utente finale. XML codifica i pacchetti dati e i pacchetti delta che rappresentano le informazioni del database. Javascript permette ai controlli HTML di interpretare e trattare i dati dei pacchetti dati XML sulla macchina client.

Per ulteriori informazioni su InternetExpress, consultare "Creazione di applicazioni Web con InternetExpress" a pagina 29-33.

Creazione di applicazioni Web Services

I Web Services sono applicazioni modulari indipendenti che possono essere diffuse e richiamate in una rete (come il World Wide Web). I Web Services dispongono di interfacce ben definite che descrivono i servizi forniti. Si utilizzano per produrre o usufruire su Internet di servizi programmabili , impiegando standard emergenti come XML, XML Schema, SOAP (Simple Object Access Protocol) e WSDL (Web Service Definition Language).

I Web Services utilizzano SOAP, un protocollo standard a basso ingombro per lo scambio di informazioni in ambiente distribuito. Utilizzano HTTP come protocollo di comunicazione e XML per codificare le chiamate a procedure remote.

È possibile utilizzare C++Builder per costruire i server che implementano i Web Services e i client che chiamano questi servizi. E' possibile scrivere dei client per server arbitrari che implementano dei Web Services che rispondono a messaggi SOAP, nonché server C++Builder che pubblicano Web Services utilizzabili da client arbitrari.

Per ulteriori informazioni sui Web Services fare riferimento al Capitolo 36, "Utilizzo dei Web Services".

Scrittura di applicazioni con COM

COM è l'acronimo di Component Object Model, un'architettura di oggetti distribuiti basata su Windows, progettata per fornire l'interoperabilità degli oggetti grazie ad una serie di routine predefinite dette interfacce. Le applicazioni COM utilizzano oggetti che vengono implementati da un processo diverso, o, se si utilizza DCOM, su una macchina diversa. È possibile anche utilizzare COM+, ActiveX e Active Server Pages.

è un modello software basato su componenti e indipendente dal linguaggio che consente l'interazione fra componenti software e applicazioni eseguiti su piattaforma Windows. L'aspetto chiave di COM è la sua capacità di consentire la comunicazione fra componenti, fra applicazioni e fra client e server tramite interfacce chiaramente definite. Le interfacce consentono ai client di chiedere a un componente COM quali funzioni supporta in fase di esecuzione. Per dotare il componente di ulteriori funzioni, è sufficiente aggiungere una opportuna interfaccia.

Utilizzo di COM e DCOM

C++Builder ha classi e wizard che semplificano la creazione degli elementi essenziali per applicazioni COM, OLE o ActiveX. È possibile creare client o server COM che implementano oggetti COM, Automation server (tra cui Active Server Objects), controlli ActiveX o ActiveForms. COM serve anche da base per altre tecnologie, come Automation, per i controlli ActiveX, Active Documents e Active Directories.

L'uso di C per creare applicazioni basate su COM offre una vasta gamma di possibilità, dalla possibilità di apportare migliorie ai progetti software, grazie all'uso delle interfacce all'interno delle applicazioni, alla creazione di oggetti in grado di interagire con altri oggetti API di sistema basati su COM, quali le estensioni alla Shell di Win9x e il supporto multimediale per DirectX. Le applicazioni possono accedere alle interfacce dei componenti COM, presenti sullo stesso computer su cui risiede l'applicazione o che esistono su un altro computer della rete, mediante un meccanismo detto Distributed COM (DCOM).

Per ulteriori informazioni su COM e sui controlli ActiveX, consultare il Capitolo 38, "Panoramica sulle tecnologie COM", Capitolo 43, "Creazione di un controllo ActiveX", e il paragrafo "Distribuzione di un'applicazione client come controllo ActiveX" a pagina 29-32.

Per ulteriori informazioni su DCOM, consultare "Uso di connessioni DCOM" a pagina 29-10.

Utilizzo di MTS e COM+

Le applicazioni COM possono essere potenziate con servizi speciali per la gestione di oggetti in un ambiente distribuito su vasta scala. Questi servizi comprendono servizi di transazione, sicurezza e gestione delle risorse forniti da *Microsoft* Transaction Server (MTS) su versioni di Windows precedenti Windows 2000), oppure COM+ (per Windows 2000 e versioni successive).

Per ulteriori informazioni su MTS e COM+, consultare il Capitolo 44, "Creazione di oggetti MTS o COM+" e il paragrafo "Uso di moduli dati transazionali" a pagina 29-7.

Utilizzo dei moduli dati

Un modulo dati può essere considerato come una speciale scheda contenente componenti non visuali. Tutti i componenti di un modulo dati *potrebbero* essere collocati su schede comuni assieme a controlli visuali. Ma se si sta progettando di riutilizzare gruppi di database e di oggetti di sistema, oppure se si desidera isolare quelle parti dell'applicazione che gestiscono la connettività al database e le regole di gestione, allora i moduli dati rappresentano un pratico strumento organizzativo.

Vi sono numerosi tipi di moduli dati, compresi quelli standard, quelli remoti, i moduli Web, i moduli applet e i servizi, a seconda della versione di C++Builder di cui si dispone. Ogni tipo di modulo dati risponde a uno scopo particolare.

- I moduli dati standard risultano particolarmente utili per applicazione database single-tier e two-tier, ma possono essere utilizzati per organizzare i componenti non visuali in qualsiasi applicazione. Per ulteriori informazioni, consultare "Creazione e modifica di moduli dati standard" a pagina 7-22.
- I moduli dati remoti formano la base di un application server nelle applicazioni database multi-tier. Essi non sono presenti in tutte le versioni. Oltre a conservare i componenti non visuali nell'application server, i moduli dati remoti espongono l'interfaccia che i client usano per comunicare con l'application server. Per ulteriori informazioni sul loro impiego, consultare "Aggiunta di un modulo dati remoto a un progetto di application server" a pagina 7-25.
- I moduli Web costituiscono la base delle applicazioni per Web server. Oltre a contenere i componenti che creano il contenuto dei messaggi di risposta HTTP, essi gestiscono la distribuzione dei messaggi HTTP provenienti dalle applicazioni client. Per ulteriori informazioni sull'utilizzo dei moduli Web, consultare il Capitolo 32, "Creazione di applicazioni server per Internet".
- I moduli Applet costituiscono la base per le applet del pannello di controllo. Oltre a conservare i controlli non visuali che implementano l'applet del pannello di controllo, definiscono le proprietà che determinano in che modo l'icona dell'applet viene visualizzata nel pannello di controllo e includono gli eventi chiamati quando gli utenti eseguono l'applet. Per maggiori informazioni sui moduli applet, consultare la Guida in linea.
- I servizi incapsulano singoli servizi in un'applicazione di servizio NT. Oltre a contenere tutti i controlli non visuali utilizzati per implementare un certo servizio, i servizi comprendono gli eventi che vengono chiamati quando il servizio viene attivato o interrotto. Per ulteriori informazioni sui servizi, consultare "Applicazioni di servizio" a pagina 7-4.

Creazione e modifica di moduli dati standard

Per creare un modulo dati standard destinato a un progetto, scegliere File | New | Data Module. C++Builder apre un modulo dati contenitore nel desktop, visualizza il file unit relativo al nuovo modulo nel Code editor, e aggiunge il modulo al progetto corrente.

In fase di progettazione, un modulo dati assomiglia a una scheda Delphi standard, con uno sfondo bianco e senza griglia di allineamento. Come nel caso delle schede, in un modulo si possono collocare componenti non visuali presi dalla Component palette e modificarne le proprietà nell'Object Inspector. È possibile ridimensionare un modulo dati per alloggiare i componenti che vi vengono aggiunti.

Facendo clic destro su un modulo si visualizza un menu contestuale. La seguente tabella riassume le opzioni del menu contestuale di un modulo dati.

Tabella 7.4 Opzioni del menu contestuale dei moduli dati

Voce di menu	Funzione
Edit	Visualizza un menu contestuale mediante il quale è possibile tagliare, copiare, incollare, cancellare e selezionare i componenti del modulo dati.
Position	Allinea i componenti non visuali alla griglia invisibile del modulo (*Align* To Grid), oppure secondo i criteri indicati nella finestra di dialogo Alignment (*Align*).
Tab Order	Permette di modificare la sequenza di accesso mediante il tasto Tab ai vari componenti e il conseguente spostamente del fuoco.
Creation Order	Permette di modificare l'ordine con cui vengono creati all'avvio i componenti di accesso ai dati.
Revert to Inherited	Abbandona le modifiche apportate a un modulo ereditato da un altro modulo nell'Object Repository, e ritorna al modulo originario ereditato.
Add to Repository	Registra nell'Object Repository un collegamento al modulo dati.
View as Text	Visualizza la rappresentazione testuale delle proprietà del modulo dati.
Text DFM	Commuta tra i formati (binario o testo) in cui viene salvato questo particolare file di scheda.

Per maggiori informazioni sui moduli dati, consultare la Guida in linea.

Attribuzione del nome a un modulo dati e al relativo file unit

La barra del titolo di un modulo dati visualizza il nome del modulo. Il nome predefinito è "DataModule*N*", in cui *N* è un numero che rappresenta il numero di unit più basso non ancora utilizzato nel progetto. Ad esempio, se si avvia un nuovo progetto, e si aggiungere ad esso un modulo prima di fare qualsiasi altra operazione, il nome predefinito del modulo dati sarà "DataModule2". Il file unit corrispondente a *DataModule2* sarà o "Unit2."

In fase di progettazione, si dovrebbe cambiare il nome ai moduli dati e ai file unit corrispondenti per renderli più descrittivi. In particolare, è bene rinominare i moduli

dati che si aggiungono all'Object Repository per evitare conflitti di nome con altri moduli dati presenti nel Repository, oppure con altre applicazioni che utilizzano i moduli.

Per rinominare un modulo dati:

1 Selezionare il modulo.

2 Modificare la proprietà *Name* del modulo nell'Object Inspector.

Non appena la proprietà *Name* nell'Object Inspector non è più evidenziata, il nuovo nome del modulo appare nella barra del titolo.

Modificando il nome di un modulo dati in progettazione se ne modifica il nome della variabile nella sezione interface del codice. Vengono modificate anche tutte le clausole uses che fanno riferimento al nome nelle dichiarazioni di procedura. È necessario invece modificare manualmente qualsiasi riferimento al modulo dati nel codice scritto in prima persona.

Per cambiare nome al file unit di un modulo dati:

1 Selezionare il file unit.

Posizionamento e assegnazione del nome ai componenti

I componenti non visuali vengono collocati in un modulo dati esattamente come i componenti visuali su una scheda. Si fa clic sul componente voluto sulla pagina appropriata della Component palette, quindi si fa clic nel modulo dati per collocare il componente. In un modulo dati, non si possono mettere controlli visuali, come le griglie. Se si tenta di farlo, viene visualizzato un messaggio di errore.

Per facilità di impiego, in un modulo dati i componenti vengono visualizzati con i loro nomi. Quando si colloca un componente, C++Builder gli assegna un nome generico che ne identifica il tipo,seguito dalla cifra *1*. Ad esempio, il componente *TDataSource* adotta il nome *DataSource1*. Questo facilita la selezione di componenti specifici di cui si desiderano gestire proprietà e metodi.

In certi casi, conviene assegnare al componente un nome diverso che ne riflette il tipo e lo scopo a cui è destinato.

Per modificare il nome di un componente in un modulo dati:

1 Selezionare il componente.
2 Modificare la proprietà *Name* del componente nell'Object Inspector.

Non appena la proprietà *Name* nell'Object Inspector non è più evidenziata, il nuovo nome del componente appare sotto la sua icona nel modulo dati.

Ad esempio, si supponga che l'applicazione database utilizzi la tabella CUSTOMER. Per accedere alla tabella, servono almeno due componenti di accesso ai dati: un componente datasource (*TDataSource)* e un componente tabella (*TClientDataSet)*. Quando si collocano questi componenti nel modulo dati, C++Builder assegna loro i nomi *DataSource1* e *ClientDataSet1*. Per riflettere sia il tipo di componente sia il database cui essi accedono, CUSTOMER, si potrebbero modificare questi nomi in *CustomerSource* e *CustomerTable*.

Utilizzo delle proprietà e degli eventi dei componenti di un modulo dati

La collocazione dei componenti in un modulo dati ne centralizza il comportamento rispetto all'intera applicazione. Ad esempio, si possono utilizzare le proprietà dei componenti dataset, come *TClientDataSet*, per controllare i dati disponibili per i componenti datasource che utilizzano questi dataset. Impostando a True la proprietà *ReadOnly* di un dataset, si impedisce agli utenti di modificare i dati visualizzati nel controllo data-aware visuale di una scheda. È possibile anche richiamare il Fields editor relativamente a un dataset facendo doppio clic su *ClientDataSet1*, per limitare i campi all'interno di una tabella o di una query a disposizione del datasource, e di conseguenza a disposizione dei controlli data-aware sulle schede. Le proprietà che vengono impostate sui componenti inclusi in un modulo dati si ripercuotono coerentemente su tutte le schede nell'applicazione che utilizzano quel modulo.

Oltre a definirne le proprietà, si possono scrivere dei gestori di evento per i componenti. Ad esempio, gli eventi che interessano un componente *TDataSource* sono tre: OnDataChange, OnStateChange e OnUpdateData. Un componente *TClientDataSet* ha più di 20 potenziali eventi.È possibile utilizzarli per creare un insieme coerente di regole di gestione che governano il trattamento dei dati in tutta l'applicazione.

Creazione di regole di gestione in un modulo dati

Oltre alla scrittura di gestori di evento per i componenti di un modulo dati, si possono codificare i metodi direttamente nel file unit di un modulo dati. Questi metodi possono essere applicati come regole di gestione alle schede che utilizzano il modulo dati. Per esempio, si supponga di scrivere una procedura che esegue la chiusura contabile mensile, trimestrale o annuale. Si può chiamare la procedura da un gestore di evento di un componente del modulo dati.

Accesso a un modulo dati da una scheda

Per associare i controlli visuali su una scheda con un modulo dati, per prima cosa è necessario aggiungere il file header del modulo dati al file .cpp della scheda. L'operazione può essere eseguita in diversi modi:

- Nell'editor di codice, aprire il file unit della scheda e includere il file header del modulo dati usando la direttiva *#include*.
- Fare clic sul file unit della scheda, scegliere il comando File|Include Unit,Hdr, quindi immettere il nome del modulo oppure sceglierlo dalla casella di riepilogo nella finestra di dialogo Use Unit.
- Per componenti database, nel modulo dati fare clic su un componente di dataset o di query per aprire il Fields editor e trascinare i campi esistenti dall'editor sulla scheda. C++Builder chiederà conferma della volontà di aggiungere il modulo alla scheda, quindi creerà i controlli (come le caselle di testo) per i campi.

Ad esempio, se si è aggiunto al modulo dati il componente *TClientDataSet*, fare doppio clic su esso per aprire il Fields editor. Selezionare un campo e trascinarlo sulla scheda. Viene visualizzato un componente casella di modifica.

Poiché il datasource non è ancora definito, C++Builder aggiunge alla scheda un nuovo componente data-source, *DataSource1,* e imposta a *DataSource1* la proprietà *DataSource* della casella di modifica. Il datasource imposta automaticamente la sua proprietà *DataSet* al componente dataset *ClientDataSet1* del modulo dati.

Aggiungendo un componente *TDataSource* al modulo dati, è possibile definire il datasource *prima* di trascinare un campo sulla scheda. Impostare la proprietà *DataSet* del datasource a *ClientDataSet1.* Dopo avere trascinato un campo sulla scheda, appare la casella di modifica con la proprietà *TDataSource* già impostata a *DataSource1.* In questo modo, il modello di accesso ai dati si presenta più "pulito".

Aggiunta di un modulo dati remoto a un progetto di application server

Alcune versioni di C++Builder consentono di aggiungere *moduli dati remoti* a progetti di application server. Un modulo dati remoto ha un'interfaccia a cui possono accedere i client di un'applicazione multi-tier attraverso la rete.

Per aggiungere un modulo dati remoto a un progetto:

1 Selezionare File | New | Other.

2 Nella finestra di dialogo New Items selezionare la pagina Multitier.

3 Fare doppio clic sull'icona Remote Data Module per aprire il wizard Remote Data Module.

Una volta aggiunto a un progetto un modulo dati remoto, è possibile utilizzarlo come un qualsiasi modulo dati standard.

Per informazioni sulle applicazioni database multi-tier, consultare il Capitolo 29, "Creazione di applicazioni multi-tier".

Utilizzo dell'Object Repository

L'Object Repository (attivato tramite il comando Tools | Repository) semplifica la condivisione di schede, di finestre di dialogo, frame e di moduli dati. Inoltre, mette a disposizione tmodelli per i nuovi progetti e wizard che guidano l'utente nella creazione di schede e progetti. Il repository è contenuto in BCB.DRO (presente per impostazione predefinita nella directory BIN), un file di testo contenente i riferimenti agli elementi che appaiono nelle finestre di dialogo Repository e New Items.

Condivisione di elementi all'interno di un progetto

È possibile condividere gli elementi *all'interno* di un progetto senza aggiungerli all'Object Repository. Quando si attiva la finestra di dialogo New Items (mediante il comando File | New | Other), è possibile vedere una pagina che riporta il nome del progetto corrente. Questa pagina elenca tutte le schede, le finestre di dialogo e i moduli dati inclusi nel progetto. È possibile derivare un nuovo elemento da un elemento esistente e personalizzarlo in base alle proprie esigenze.

Aggiunta di elementi all'Object Repository

È possibile aggiungere i propri progetti, le proprie schede, riquadri e i propri moduli dati a quelli già disponibili nell'Object Repository. Per aggiungere un elemento all'Object Repository,

1. Nel caso l'elemento sia un progetto o contenuto in un progetto, aprire il progetto.
2. Nel caso di un progetto, scegliere il comando Project | Add To Repository. Nel caso sia una scheda o un modulo dati, fare clic destro sull'elemento e scegliere il comando Add To Repository.
3. Immettere una descrizione, un titolo e l'autore.
4. Decidere su quale pagina della finestra di dialogo New Items si vuole che appaia l'elemento, quindi scrivere il nome della pagina oppure selezionarla dalla casella combinata Page. Se si scrive il nome di una pagina inesistente, C++Builder creerà una nuova pagina.
5. Scegliere Browse per selezionare un'icona che rappresenterà l'oggetto nell'Object Repository.
6. Scegliere OK.

Condivisione di oggetti in un team di lavoro

È possibile condividere oggetti all'interno del proprio gruppo di lavoro o di sviluppo rendendo disponibile in rete il repository. Per utilizzare un repository condiviso, tutti i membri del team devono indicare per l'opzione Shared Repository della finestra di dialogo Environment Options la stessa directory:

1. Scegliere il comando Tools | Environment Options.
2. Sulla pagina Preferences, individuare il pannello Shared Repository. Nella casella di testo Directory, immettere la directory in cui si desidera ubicare il repository condiviso. Accertarsi di specificare una directory accessibile a tutti i membri del team.

La prima volta che si aggiungerà un elemento al repository, C++Builder creerà nella directory indicata per lo Shared Repository un file di nome BCB.DRO, nel caso non ne esista già uno.

Utilizzo di un elemento dell'Object Repository in un progetto

Per accedere agli elementi nell'Object Repository, scegliere il comando File | New | Other. Apparirà la finestra di dialogo New Items che mostra tutti gli elementi disponibili. In base al tipo di elemento che si desidera utilizzare, saranno disponibili fino a tre opzioni per aggiungere l'elemento al progetto:

- Copy
- Inherit
- Use

Copia di un elemento

Scegliere l'opzione Copy per fare una copia esatta dell'elemento scelto e aggiungere la copia al progetto. Le successive modifiche apportate all'elemento nell'Object Repository non influenzeranno la copia, come pure le modifiche apportate alla copia non influenzeranno l'elemento originale nell' Object Repository.

L'opzione Copy è l'unica disponibile per modelli di progetto.

Ereditare un elemento

Scegliere l'opzione Inherit per derivare una nuova classe dall'elemento scelto nell'Object Repository e aggiungere la nuova classe al progetto. Quando si ricompila il progetto, qualsiasi modifica apportata all'elemento nell'Object Repository si ripercuoterà sulla classe derivata, in aggiunta alle modifiche apportate all'elemento nel progetto. Le modifiche apportate alla classe derivata non si ripercuotono sull'elemento condiviso dell'Object Repository.

L'opzione Inherit è disponibile per schede, finestre di dialogo e moduli dati ma non per modelli di progetto. È l'*unica* opzione disponibile per riutilizzare elementi all'interno dello stesso progetto.

Utilizzo di un elemento

Scegliere l'opzione Use quando si desidera che l'elemento scelto diventi parte del progetto. Le modifiche apportate all'elemento nel progetto appariranno in tutti gli altri progetti a cui è stato aggiunto l'elemento mediante le opzioni Inherit o Use. Selezionare questa opzione con estrema attenzione.

L'opzione Use è disponibile per schede, finestre di dialogo e moduli dati.

Utilizzo di modelli di progetto

I modelli (template) sono progetti predefiniti che è possibile utilizzare come punto di partenza per le proprie applicazioni. Per creare un nuovo progetto a partire da un modello:

1. Scegliere il comando File|New|Other per visualizzare la finestra di dialogo New Items.
2. Scegliere la pagina Projects.
3. Selezionare il modello di progetto desiderato e scegliere OK.
4. Nella finestra di dialogo Select Directory, specificare una directory per i file del nuovo progetto.

C++Builder copia i file di modello nella directory specificata, in cui sarà possibile modificarli. Il modello originale del progetto non sarà influenzato dalle eventuali modifiche.

Modifica di elementi condivisi

Se si modifica un elemento nell'Object Repository, le modifiche si ripercuoteranno su tutti i successivi progetti che utilizzano l'elemento oltre che sui progetti esistenti a cui è stato aggiunto l'elemento mediante le opzioni Use o Inherit. Per evitare la propagazione di modifiche agli altri progetti, esistono varie alternative:

- Copiare l'elemento e modificarlo solo nel progetto corrente.
- Copiare l'elemento nel progetto corrente, modificarlo, quindi aggiungerlo al Repository con un nome differente.
- A partire dall'elemento, creare un componente, una DLL, un modello di componente o un frame. Se si crea un componente o una DLL, è possibile condividerli con gli altri sviluppatori.

Specifica di un progetto predefinito, di una nuova scheda e della scheda principale

Per impostazione predefinita, quando si sceglie il comando File|New|Application o File|New|Form, C++Builder visualizza una scheda vuota. È possibile modificare questo comportamento riconfigurando il Repository:

1 Scegliere il comando Tools|Repository.

2 Se si desidera specificare un progetto predefinito, selezionare la pagina Projects e scegliere un elemento in Objects. Quindi selezionare la casella di controllo New Project.

3 If you want to specify a default form, select a Repository page (such as Forms), them choose a form under Objects. To specify the default new form (File|New|Form), select the New Form check box. To specify the default main form for new projects, select the Main Form check box.

4 Fare clic su OK.

Attivazione dell'Help nelle applicazioni

Sia la VCL che CLX hanno la capacità di visualizzare l'Help delle applicazioni tramite un meccanismo basato su oggetti che permette l'inoltro delle richieste di Help a uno dei tanti visualizzatori di Help esterni. Per supportare questo sistema, un'applicazione deve includere una classe che implementa l'interfaccia *ICustomHelpViewer* (e, facoltativamente, una delle numerose interfacce da essa derivate), e quindi registrare sé stessa mediante l'Help Manager globale.

La VCL fornisce a tutte le applicazioni un'istanza di *TWinHelpViewer*, che implementa tutte queste interfacce e fornisce il collegamento tra le applicazioni e WinHelp. CLX richiede che venga fornita una propria implementazione. In Windows, le applicazioni CLX possono utilizzare la unit WinHelpViewer, fornita come parte della VCL, se si collegano ad esso in modo statico—cioè, includendo quella unit come parte del progetto invece di collegarla con il package della VCL.

L'Help Manager gestisce una lista di visualizzatori registrati e passa loro le richieste in un processo bifase: prima chiede a ogni visualizzatore se può fornire supporto per una particolare parola chiave dell'Help o contesto e quindi passa la richiesta Help al visualizzatore che ha risposto che può fornire tale supporto.

Se esiste più di un visualizzatore che supporta la parola chiave, come nel caso di un'applicazione che ha registrato visualizzatori sia per WinHelp e per HYperHelp su Windows oppure sia per Man sia per Info su Linux, l'Help Manager può visualizzare un riquadro di selezione attraverso cui l'utente dell'applicazione può determinare quale visualizzatore di Help richiamare. Altrimenti, visualizza il primo sistema di Help che è stato rilevato.

Interfacce per il sistema di Help

Il sistema di Help consente la comunicazione fra l'applicazione e i visualizzatori di Help mediante una serie di interfacce. Queste interfacce sono tutte definite in HelpIntfs, che contiene anche l'implementazione dell'Help Manager.

ICustomHelpViewer fornisce il supporto per la visualizzazione dell'Help in base a una parola chiave fornita, e per la visualizzazione di un indice che elenca tutti i file di Help disponibili in un particolare visualizzatore.

IExtendedHelpViewer fornisce il supporto per la visualizzazione dell'Help in base a un contesto di Help numerico e per la visualizzazione di argomenti; nella maggior parte dei sistemi di Help, gli argomenti fungono da parole chiave ad alto livello (ad esempio, "IntToStr" potrebbe essere una parola chiave nel sistema di Help, mentre "String manipulation routines" potrebbe essere il nome di un argomento).

*ISpecialWinHelpViewe*r fornisce il supporto per la risposta a messaggi specializzati di WinHelp che un'applicazione in esecuzione in ambiente Windows potrebbe ricevere e che non sono facilmente generalizzabili. Di solito, solo le applicazioni che operano in ambiente Windows hanno la necessità di implementare questa interfaccia, e anche in questo caso l'implementazione è necessaria solo per le applicazioni che fanno un uso intensivo di messaggi non-standard di WinHelp.

IHelpManager fornisce un meccanismo che consente al visualizzatore di Help di dare una risposta allo Help Manager dell'applicazione e di richiedere ulteriori informazioni. Uno *IHelpManager* viene ottenuto nel momento in cui il visualizzatore Help si registra.

IHelpSystem fornisce un meccanismo attraverso il quale *TApplication* passa le richiede di Help al sistema di Help. *TApplication* ottiene un'istanza di un oggetto che implementa sia *IHelpSystem* sia *IHelpManager* ed esporta queste istanze sotto forma di proprietà; ciò consente ad altro codice all'interno dell'applicazione di presentare direttamente, quando occorre, le richieste di Help.

IHelpSelector fornisce un meccanismo attraverso cui il sistema di Help può richiamare l'interfaccia utente per domandare quale visualizzatore di Help dovrebbe essere utilizzato, nei casi in cui vi sia più di un visualizzatore in gradi di gestire una richiesta di Help, e per visualizzare un Indice generale. Questa capacità di visualizzazione non è incorporata direttamente nello Help Manager per poter avere

un unico codice di Help Manager, indipendentemente da quale set di widget o libreria di classi si utilizzi.

Implementazione di ICustomHelpViewer

L'interfaccia *ICustomHelpViewer* contiene tre tipi di metodi: metodi utilizzati per comunicare con l'Help Manager informazioni a livello di sistema (ad esempio, informazioni non correlate a una particolare richiesta di Help); metodi relativi alla visualizzazione dell'Help in base a una parola chiave fornita dall'Help Manager; e metodi per la visualizzazione di un indice.

Comunicazione con Help Manager

ICustomHelpViewer fornisce tre funzioni che possono essere utilizzate per comunicare informazioni di sistema con l'Help Manager:

- *GetViewerName*
- *NotifyID*
- *ShutDown*
- *SoftShutDown*

L'HelpManager chiama queste funzioni nelle seguenti circostanze:

- AnsiString *ICustomHelpViewer::GetViewerName*() viene chiamata quando l'Help Manager vuole conoscere il nome del visualizzatore (ad esempio, se all'applicazione viene chiesto di visualizzare una lista di tutti i visualizzatori registrati). Queste informazioni vengono restituite per mezzo di una stringa, che deve ovviamente essere di tipo statico, (cioè, non può cambiare durante l'operazione dell'applicazione). I set di caratteri multibyte non sono supportati.
- **void** *ICustomHelpViewer::NotifyID*(**const** int ViewerID) viene chiamata **immediatamente** dopo la registrazione in modo da fornire al visualizzatore un cookie univoco che la identifichi. Queste informazioni devono essere memorizzate per un successivo utilizzo; se il visualizzatore si chiude autonomamente (invece che in risposta a una notifica da parte dell'Help Manager), deve fornire all'Help Manager il cookie di identificazione, in modo che l'Help Manager possa rilasciare tutti i riferimenti al visualizzatore. (Il non fornire il cookie o il fornirne uno errato, fa sì che l'Help Manager potenzialmente rilasci i riferimenti al visualizzatore errato.)
- **void** *ICustomHelpViewer::ShutDown*() viene chiamato dall'Help Manager per chiedere al visualizzatore di Help di chiudere qualsiasi manifestazione esternamente visibile del sistema di Help (ad esempio, delle finestre che visualizzano informazioni della guida) senza però scaricare il visualizzatore.
- **void** *ICustomHelpViewer::SoftShutDown*() viene chiamata dall'Help Manager per informare il visualizzatore di Help che il Manager sta per essere chiuso e che tutte le risorse allocate dal visualizzatore di Help dovrebbero essere liberate. Si consiglia di delegare a questo metodo tutte le risorse da liberare.

Richiesta di informazioni all'Help Manager

I visualizzatori di Help sono in comunicazione con l'Help Manager attraverso l'interfaccia *IHelpManager*, una cui istanza viene loro restituita quando vengono registrati con l'Help Manager. *IHelpManager* permette al visualizzatore Help di comunicare quattro cose:

- Una richiesta dello handle di finestra del controllo attualmente attivo.
- Una richiesta del nome del file di Help che l'Help Manager crede dovrebbe contenere la guida relativa al controllo attivo al momento.
- Una richiesta del percorso per quel file di Help.
- Una notifica che il visualizzatore Help si sta chiudendo in risposta a qualcosa diverso da una richiesta dall'Help Manager in tale senso.

int *IHelpManager::GetHandle*() viene chiamata dal visualizzatore di Help nel caso abbia bisogno di conoscere lo handle del controllo attualmente attivo; il risultato è un handle di finestra.

AnsiString *IHelpManager::GetHandle*() viene chiamata dal visualizzatore di Help nel caso desideri conoscere il nome del file di Help che, stando al controllo corrente, dovrebbe contenere l'aiuto per il controllo.

void *IHelpManager::Release*() viene chiamata per notificare all'Help Manager che un visualizzatore di Help si sta scollegando. Non dovrebbe essere mai chiamata in risposta a una richiesta proveniente da *IHelpManager::ShutDown*(); viene utilizzata solo per notificare all'Help Manager di una sconnessione inaspettata.

Visualizzazione di Help in base a parole chiave

Di solito le richieste di aiuto arrivano dal visualizzatore Help o sotto forma di aiuto in base a una parola chiave, nel qual caso al visualizzatore viene richiesto di fornire un aiuto in base a una particolare stringa, oppure come aiuto in base al contesto, nel qual caso al visualizzatore viene richiesto di fornire un aiuto in base a un particolare identificativo numerico.

I contesti numerici di aiuto sono la forma standard delle richieste di Help nelle applicazioni eseguite in ambiente Windows che utilizzano il sistema WinHelp; benché supportate dalla CLX, se ne sconsiglia l'utilizzo nelle applicazioni CLX perché la maggior parte dei sistemi di Help di Linux non sono in grado di comprenderle.

Alle implementazioni di *ICustomHelpViewer* viene richiesto di fornire il supporto per richieste di aiuto in base a una parola chiave, mentre alle implementazioni *IExtendedHelpViewer* viene richiesto il supporto per le richieste di aiuto in base al contesto.

ICustomHelpViewer fornisce tre metodi per la gestione di aiuto in base a una parola chiave:

- *UnderstandsKeyword*
- *GetHelpStrings*
- *ShowHelp*

```
int__fastcall ICustomHelpViewer::UnderstandsKeyword(const AnsiString HelpString)
```

è il primo dei tre metodi chiamati dall'Help Manager, il quale chiamerà ogni visualizzatore di Help registrato passandogli la stessa stringa per chiedergli se è in grado di fornire un aiuto per quella stringa; il visualizzatore dovrebbe risponde con un valore integer che indica quante differenti pagine della guida può visualizzare in risposta a quella richiesta di Help. Per determinare tale valore, il visualizzatore può utilizzare un qualsiasi metodo- all'interno dell'IDE il visualizzatore HyperHelp gestisce il proprio indice e lo utilizza per la ricerca. Se il visualizzatore non è in grado di fornire un aiuto per questa parola chiave dovrebbe restituire zero. I numeri negativi vengono interpretati come se fossero zero, ma questo comportamento non è garantito nelle versioni successive.

```
Classes::TStringList*__fastcall ICustomHelpViewer::GetHelpStrings(const AnsiString
HelpString)
```

viene chiamato dall'Help Manager nel caso vi sia più di un visualizzatore in grado di fornire un aiuto su un argomento. Il visualizzatore dovrebbe restituire uno *TStringList* liberato dall'Help Manager. Le stringhe nella lista restituita dovrebbero corrispondere alle pagine disponibili per quella parola chiave, ma le caratteristiche di quella mappatura possono essere determinate dal visualizzatore. Nel caso del visualizzatore WinHelp in Windows e del visualizzatore HyperHelp in Linux, l'elenco di stringhe contiene sempre esattamente una voce. HyperHelp fornisce una propria indicizzazione e la sua duplicazione in un'altra posizione produrrebbe una duplicazione senza puntatore. Nel caso del visualizzatore Man page (Linux), la lista di stringhe è formata da stringhe multiple, una per ogni sezione del manuale che contiene una pagina per quella parola chiave.

```
void__fastcall ICustomHelpViewer::ShowHelp(const AnsiString HelpString)
```

viene chiamato dall'Help Manager nel caso abbia bisogno che il visualizzatore di Help visualizzi un aiuto per una particolare parola chiave. Questa è l'ultima chiamata a metodo dell'operazione; il sistema garantisce che non sarà mai chiamato a meno che prima non venga chiamato il metodo *UnderstandsKeyword*.

Visualizzazione degli indici

ICustomHelpViewer fornisce due metodi relativi alla visualizzazione degli indici:

- *CanShowTableOfContents*
- *ShowTableOfContents*

Il meccanismo del loro funzionamento è simile a quello delle operazioni di richiesta di help in base a una parola chiave: per prima cosa l'Help Manager interroga tutti i visualizzatori di Help chiamando *ICustomHelpViewer::CanShowTableOfContents()* e successivamente richiama un particolare visualizzatore di Help chiamando *ICustomHelpViewer::ShowTableOfContents*().

È perfettamente normale che un particolare visualizzatore si rifiuti di supportare richieste per la visualizzazione di un indice. Questo è il caso, ad esempio, del visualizzatore Man page, perché il concetto di indice non si addice alla modalità di funzionamento di Man page; il visualizzatore HyperHelp, invece, supporta un indice passando la richiesta di visualizzare l'indice direttamente a HyperHelp su Linux e a

WinHelp su Windows. *Non* è normale, tuttavia, che un'implementazione di *ICustomHelpViewer* risponda a una richiesta che perviene da *CanShowTableOfContents* con **true** e poi ignori le richieste che pervengono da *ShowTableOfContents*.

Implementazione IExtendedHelpViewer

ICustomHelpViewer fornisce solo un supporto diretto per Help basati su parole chiave. Alcuni sistemi di Help (specialmente WinHelp) funzionano associando numeri (conosciuti come ID di contesto) a parole chiave in un modo interno al sistema di Help e quindi non visibile all'applicazione. Tali sistemi richiedono che l'applicazione supporti l'Help in base al contesto, vale a dire l'applicazione richiama il sistema Help con quel contesto, invece che con una stringa, e il sistema di Help traduca tale numero.

Le applicazioni scritte nella CLX e nella CLX possono dialogare con sistemi che richiedono un Help in base al contesto estendendo l'oggetto che implementa *ICustomHelpViewer* in modo che implementi anche *IExtendedHelpViewer. IExtendedHelpViewer* fornisce anche un supporto per dialogare con sistemi di Help che permettono di passare direttamente ad argomenti di livello più alto invece di utilizzare ricerche per parole chiave. Il visualizzatore natico WinHelp compie l'operazione automaticamente.

IExtendedHelpViewer espone quattro funzioni. Due di esse - *UnderstandsContext* e *DisplayHelpByContext* - sono utilizzate per supportare gli Help in base al contesto; le altre due - *UnderstandsTopic* and *DisplayTopic* - sono utilizzate per supportare gli argomenti.

Quando un utente dell'applicazione preme il tasto F1, l'Help Manager chiama

```
int__fastcall IExtendedHelpViewer::UnderstandsContext(const int ContextID, AnsiString
HelpFileName)
```

e il controllo attivato attualmente supporta un Help in base al contesto invece che basato su parola chiave. Come nel caso di *ICustomHelpViewer::UnderstandsKeyword*, l'Help Manager interroga in sequenza tutti i visualizzatori di Help registrati. A differenza del caso con *ICustomHelpViewer::UnderstandsKeyword*(), però, se più di un visualizzatore supporta un determinato contesto, viene attivato il primo visualizzatore registrato che supporta quel certo contesto.

L'Help Manager chiama

```
void__fastcall IExtendedHelpViewer::DisplayHelpByContext(const int ContextID, AnsiString
HelpFileName)
```

dopo aver eseguito la scansione ciclica dei visualizzatori di Help registrati.

Le funzioni per il supporto degli argomenti operano in modo analogo:

```
bool__fastcall IExtendedHelpViewer::UnderstandsTopic(const AnsiString Topic)
```

viene utilizzata per eseguire la scansione ciclica dei visualizzatori di Help e chiedere se supportano un argomento;

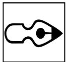
```
void__fastcall IExtendedHelpViewer::DisplayTopic(const AnsiString Topic)
```

viene utilizzata per richiamare il primo visualizzatore registrato che riferisce di essere in grado di fornire l'aiuto per quell'argomento.

Implementazione IHelpSelector

IHelpSelector viene utilizzato in combinazione con *ICustomHelpViewer*. Quando esistono più visualizzatori di Help che sostengono di essere in grado di fornire un supporto per una certa parola chiave, per un contesto o per un determinato argomento, o di essere in grado di fornire un indice, l'Help Manager deve effettuare una scelta. Nel caso di contesti o di argomenti, l'Help Manager seleziona sempre il primo visualizzatore di Help che sostiene di essere in grado di fornire il supporto. Nel caso delle parole chiave o dell'indice, per impostazione predefinita l'Help Manager selezionerà il primo visualizzatore Help. Questo comportamento può essere ridefinito da un'applicazione.

Per ridefinire la decisione dell'Help Manager in questi casi, l'applicazione deve registrare una classe che fornisca un'implementazione dell'interfaccia *IHelpSelector.IHelpSelector* esporta due funzioni: *SelectKeyword*, e *TableOfContents.* Entrambe accettano come argomento un *TStrings* contenente, ad uno ad uno, o le possibili corrispondenze con la parola chiave o i nomi dei visualizzatori che sostengono di essere in grado di fornire un indice. Chi implementa la funzione deve restituire l'indice (in *TStringList*) che rappresenta la stringa selezionata, *TStringList* viene quindi liberata dall'Help Manager.



L'Help Manager potrebbe fare confusione se le stringhe vengono riorganizzate; si consiglia a chi implementa *IHelpSelector* di astenersi dal fare ciò. Il sistema di Help supporta *un solo* HelpSelector; quando si registrano nuovi selettori, qualsiasi selettore precedentemente esistente viene scollegato.

Registrazione di oggetti di sistema di Help

Affinché l'Help Manager possa comunicare con loro, gli oggetti che implementano *ICustomHelpViewer*, *IExtendedHelpViewer*, ISpecialWinHelpViewer e *IHelpSelector* devono essere registrati con l'Help Manager.

Per registrare con l'Help Manager oggetti di sistema di Help, è necessario:

- Registrare il visualizzatore di Help.
- Registrare il selettore di Help.

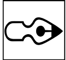
Registrazione dei visualizzatori di Help

La unit che contiene l'implementazione di oggetto deve utilizzare HelpIntfs. Nel file header della unit che lo implementa si deve dichiarare un'istanza dell'oggetto.

La unit implementante deve includere una direttiva di avvio pragma che chiama un metodo che assegna la variabile di istanza e la passa alla funzione *RegisterViewer. RegisterViewer* è una funzione esportata da HelpIntfs.pas che accetta come argomento uno *ICustomHelpViewer* e restituisce uno *IHelpManager.* Lo *IHelpManager* dovrebbe essere memorizzato per un successivo utilizzo.

Il file .cpp corrispondente contiene il codice per registrare l'interfaccia. Per l'interfaccia descritta sopra, il codice di registrazione assomiglia al seguente:

```
void InitServices()
{
  THelpImplementor GlobalClass;
  Global = dynamic_cast<ICustomHelpViewer*>(GlobalClass);
  Global->AddRef;
  HelpIntfs::RegisterViewer(Global, GlobalClass->Manager);
}
#pragma startup InitServices
```



L'oggetto Help Manager deve essere liberato nel distruttore dell'oggetto GlobalClass nel caso non sia stato già liberato.

Registrazione dei selettori di Help

La unit che contiene l'implementazione dell'oggetto deve utilizzare Forms nella VCL oppure QForms Nella CLX. Nel file .ccp dell'unit che lo implementa si deve dichiarare un'istanza dell'oggetto.

La unit che lo implementa deve registrare il selettore di Help attraverso la proprietà *HelpSystem* dell'oggetto globale Application:

```
Application->HelpSystem->AssignHelpSelector(myHelpSelectorInstance)
```

Questa procedura non restituisce un valore.

Utilizzo dell'Help in un'applicazione VCL

Le seguenti sezioni spiegano come utilizzare l'Help in un'applicazione VCL.

- In che modo TApplication elabora l'Help VCL
- In che modo i controlli VCL elaborano l'Help
- Chiamata diretta a un sistema di Help
- Utilizzo di IHelpSystem

In che modo TApplication elabora l'Help VCL

TApplication della VCL fornisce due metodi accessibili dal codice dell'applicazione:

Tabella 7.5 Metodi Help in TApplication

HelpCommand	Accetta un HELP_COMMAND in stile Windows Help e lo passa a WinHelp. Le richieste di Help inviate attraverso questo meccanismo vengono passate solo alle implementazioni di IspecialWinHelpViewer.
HelpContext	Richiama il sistema di Help con una richiesta di Help in base al contesto.
HelpKeyword	Richiama il sistema di Help con una richiesta di Help in base a una parola chiave.
HelpJump	Richiede la visualizzazione di un particolare argomento.

Tutte e quattro le funzioni prendono i dati che vengono loro passati e li inoltrano tramite un membro dati di *TApplication* che rappresenta il sistema di Help. Quel membro dati è direttamente accessibile attraverso la proprietà *HelpSystem* a sola lettura.

In che modo i controlli VCL elaborano l'Help

Tutti i controlli della VCL che derivano da *TControl* espongono diverse proprietà utilizzate dal sistema di Help: *HelpType*, *HelpContext*, e *HelpKeyword*.

La proprietà *HelpType* contiene un'istanza di tipo enumerativo che determina se chi ha progettato il controllo si aspettava che l'aiuto fosse fornito in base a una parola chiave o in base al contesto. Se *HelpType* è impostata a *htKeyword*, il sistema Help si aspetta che il controllo utilizzi un help basato su parole chiave e quindi controllerà solo il contenuto della proprietà *HelpKeyword*. Al contrario, se *HelpType* è impostato a *htContext*, il sistema Help si aspetta che il controllo utilizzi un Help basato sul contesto e controllerà solo il contenuto della proprietà *HelpContext*.

Oltre alle proprietà, i controlli espongono un singolo metodo, *InvokeHelp*, che può essere chiamato per passare una richiesta al sistema di Help. Tale metodo non richiede parametri e chiama i metodi nell'oggetto globale Application che corrispondono al tipo di help supportato dal controllo.

I messaggi di help vengono richiamati automaticamente quando si preme F1 perché il metodo *KeyDown* di *TWinControl* richiama *InvokeHelp*.

Utilizzo dell'Help in un'applicazione CLX

Le seguenti sezioni spiegano come utilizzare l'Help in un'applicazione CLX.

- In che modo TApplication elabora l'Help CLX
- In che modo i controlli CLX elaborano l'Help
- Chiamata diretta a un sistema di Help
- Utilizzo di IHelpSystem

In che modo TApplication elabora l'Help CLX

TApplication della CLX fornisce due metodi accessibili dal codice dell'applicazione:

- *ContextHelp*, che richiama il sistema di Help con una richiesta di Help in base al contesto
- *KeywordHelp*, che richiama il sistema di Help con una richiesta di Help in base a una parola chiave

Entrambe le funzioni accettano come argomento il contesto o la parola chiave che viene loro passato e inviano la richiesta attraverso un membro dati di *TApplication*, che rappresenta il sistema di Help. Quel membro dati è direttamente accessibile attraverso la proprietà *HelpSystem* sola lettura.

In che modo i controlli CLX elaborano l'Help

Tutti i controlli che derivano da *TControl* espongono quattro proprietà utilizzate dal sistema di Help: *HelpType*, *HelpFile*, *HelpContext*, e *HelpKeyword*. *HelpFile* dovrebbe contenere il nome del file in cui è situato l'aiuto per il controllo; se l'aiuto è situato in un sistema di Help esterno che non ha bisogno di nomi di file (ad esempio, il sistema Man page), non si dovrebbe impostare un valore per la proprietà.

La proprietà *HelpType* contiene un'istanza di tipo enumerativo che determina se chi ha progettato il controllo si aspettava che l'aiuto fosse fornito in base a una parola chiave o in base al contesto; le altre due proprietà sono collegate a quest'ultima. Se *HelpType* è impostata a *htKeyword*, il sistema Help si aspetta che il controllo utilizzi un help basato su parole chiave e quindi controllerà solo il contenuto della proprietà *HelpKeyword*. Al contrario, se *HelpType* è impostato a *htContext*, il sistema Help si aspetta che il controllo utilizzi un Help basato sul contesto e controllerà solo il contenuto della proprietà *HelpContext*.

Oltre alle proprietà, i controlli espongono un singolo metodo, *InvokeHelp*, che può essere chiamato per passare una richiesta al sistema di Help. Tale metodo non richiede parametri e chiama i metodi nell'oggetto globale Application che corrispondono al tipo di aiuto supportato dal controllo.

I messaggi di aiuto vengono richiamati automaticamente quando si preme F1 perché il metodo *KeyDown* di *TWidgetControl* richiama *InvokeHelp*.

Chiamata diretta a un sistema di Help

Per ulteriori funzionalità di sistema di Help non fornite dalla VCL o dalla CLX, *TApplication* fornisce una proprietà a sola lettura che consente di accedere direttamente al sistema di Help. Questa proprietà è un'istanza di un'implementazione dell'interfaccia *IHelpSystem*. *IHelpSystem* e *IHelpManager* sono implementati dallo stesso oggetto, ma un'interfaccia viene utilizzata per consentire all'applicazione di dialogare con l'Help Manager, mentre l'altra viene utilizzata per consentire ai visualizzatori di Help di dialogare con l'Help Manager.

Utilizzo di IHelpSystem

IHelpSystem permette alle applicazioni VCL o CLX di fare tre cose:

- Fornisce all'Help Manager le informazioni sui percorsi.
- Fornisce un nuovo selettore di Help.
- Chiede all'Help Manager di visualizzare l'Help.

L'assegnazione di un selettore di Help consente all'Help Manager di delegare il processo di decisione in quei casi in cui più sistemi di Help esterni possono fornire un help per la stessa parola chiave. Per ulteriori informazioni, vedere la sezione "Implementazione IHelpSelector" on page 7-34.

IHelpSystem esporta quattro procedure e una funzione per richiedere all'Help Manager di visualizzare l'help:

- *ShowHelp*
- *ShowContextHelp*
- *ShowTopicHelp*
- *ShowTableOfContents*
- *Hook*

Hook è destinata esclusivamente alla compatibilità con WinHelp e non dovrebbe essere utilizzata in un'applicazione CLX; permette l'elaborazione di messaggi WM_HELP che non possono essere mappati direttamente su richieste di Help basate su parola chiave, su contesto o su argomento. Ognuno degli altri metodi richiede due argomenti: la parola chiave, l'ID di contesto o l'argomento per cui si chiede aiuto e il file di Help in cui ci si aspetta di trovare l'aiuto.

Di solito, a meno che non si stia richiedendo aiuto in base a un argomento, è ugualmente efficiente e molto più chiaro passare le richieste di aiuto all'Help Manager tramite il metodo *InvokeHelp* del controllo.

Personalizzazione del sistema di Help dell'IDE

L'IDE di C++Builder supporta più visualizzatori di Help utilizzando esattamente lo stesso meccanismo delle applicazioni VCL e CLX: delega le richieste di Help all'Help Manager, il quale le invia ai visualizzatori di Help registrati. L'IDE sfrutta il medesimo WinHelpViewer utilizzato da VCL.

Per installare nell'IDE un nuovo visualizzatore di Help, si fa esattamente ciò che si farebbe in un'applicazione CLX, con una sola differenza. Si scrive un oggetto che implementa *ICustomHelpViewer* (e, se occorre, *IExtendedHelpViewer*) per inviare le richieste di Help al visualizzatore esterno preferito e si registra *ICustomHelpViewer* con l'IDE.

Per registrare con l'IDE un visualizzatore custom di Help:

1. Accertarsi che la unit che implementa il visualizzatore di Help contenga HelpIntfs.cpp.
2. Eseguire il build della unit in un package di progettazione registrato con l'IDE ed eseguire il build del package attivando l'opzione relativa ai package di esecuzione. (Questo è necessario per essere certi che l'istanza dell'Help Manager utilizzata dalla unit sia la stessa dell'istanza dell'Help Manager utilizzata dall'IDE.)
3. Accertarsi che il visualizzatore di Help esista come istanza globale all'interno della unit.
4. Nella funzione initialization bloccata da à #,pragma startup, accertarsi che l'istanza venga passata alla funzione *RegisterHelpViewer.*

Capitolo

8

Sviluppo dell'interfaccia utente dell'applicazione

Quando si apre C++Builder o si crea un nuovo progetto, sullo schermo viene visualizzata una scheda vuota. Si progetta l'interfaccia utente di applicazione (UI) collocando e organizzando sulla scheda i componenti visuali, come finestre, menu e finestre di dialogo, prelevati dalla Component palette.

Quando il componente visuale è sulla scheda, è possibile regolarne la posizione, le dimensioni e le altre proprietà regolabili in fase di progettazione e codificare i gestori di evento. C++Builder si occupa dei dettagli sottostanti della programmazione.

Le seguenti sezioni descrivono alcune delle principali attività sull'interfaccia, come le operazioni sulle schede, la creazione di modelli di componenti, l'aggiunta di finestre di dialogo e l'organizzazione di azioni per i menu e per le barre strumenti.

Controllo del comportamento dell'applicazione

TApplication, *TScreen* e *TForm* sono classi the che costituiscono la struttura portante di tutte le applicazioni C++Builder e controllano il comportamento del progetto. La classe *TApplication* costituisce la base di un'applicazione, in quanto fornisce le proprietà e i metodi che incapsulano il comportamento di un programma standard. *TScreen* viene usata in fase di esecuzione per tenere traccia delle schede e dei moduli dati che sono stati caricati, nonché delle informazioni specifiche di sistema, come la risoluzione dello schermo e i font video disponibili. Le istanze della classe *TForm* sono i blocchi per la costruzione dell'interfaccia utente dell'applicazione. Le finestre dell'applicazione, comprese quelle di dialogo, sono basate su *TForm*.

Il lavoro a livello di applicazione

La variabile globale *Application*, di tipo *TApplication*, si trova in ogni applicazione basata su VCL o su CLX. *Application* incapsula l'applicazione e fornisce molte funzioni che si verificano nel background del programma. Ad esempio, *Application* gestirà il modo in cui viene chiamato un file Help dal menu del programma. La comprensione di come funziona *TApplication* è molto importante per uno scrittore di componenti e per gli sviluppatori di applicazioni autonome; ma, quando si crea un progetto, è bene anche capire come impostare le opzioni che *Application* gestisce nella pagina Project| Options *Application*.

Inoltre, *Application* riceve moltissimi eventi che si riferiscono all'intera applicazione. Ad esempio, l'evento *OnActivate* consente di eseguire azioni non appena si avvia l'applicazione, l'evento *OnIdle* consente di eseguire processi in background quando l'applicazione non è occupata, l'evento *OnMessage* consente di intercettare i messagi di Windows (solo in ambiente Windows), l'evento *OnEvent* consente di intercettare gli eventi, e così via. Sebbene non sia possibile utilizzare l'IDE per esaminare le proprietà e gli eventi della variabile globale *Application*, esiste un altro componente, *TApplicationEvents*, che intercetta gli eventi e consente di preparare i gestori di evento utilizzando l'IDE.

Gestione dello schermo

Quando si crea un progetto, viene creata una variabile globale di tipo *TScreen* denominata *Screen*. *Screen* incapsula lo stato dello schermo sul quale viene eseguita l'applicazione. Le normali operazioni eseguite da *Screen* includono la specifica:

- dell'aspetto del cursore.
- delle dimensioni della finestra nella quale viene eseguita l'applicazione.
- dell'elenco dei font disponibili per lo schermo.
- il comportamento su più schermi (solo Windows).

Se l'applicazione Windows viene eseguita su più monitor, *Screen* conserva l'elenco dei monitor e le loro dimensioni, in modo che si possa gestire efficacemente il layout dell'interfaccia utente.

Se si utilizza CLX per programmi multipiataforma, il comportamento predefinito delle applicazioni è quello di creare un componente screen in base alle informazioni sul dispositivo video corrente e di assegnarlo a *Screen*.

Impostazione delle schede

TForm è la classe fondamentale per la creazione di applicazioni GUI. Quando si apre C++Builder e si visualizza un progetto predefinito o si crea un nuovo progetto, viene visualizzata automaticamente una scheda che consente di iniziare il progetto dell'interfaccia utente.

Uso della scheda principale

La prima scheda creata e salvata in un progetto diventa, per impostazione predefinita, la scheda principale del progetto, ovvero la prima scheda creata in fase di esecuzione. Quando poi si aggiungono schede ai progetti, si può decidere di designare un'altra scheda come scheda principale dell'applicazione. Inoltre, la specificazione di una scheda come principale è un modo facile per verificarla in fase di esecuzione, in quanto, a meno che non si cambia l'ordine di creazione della scheda, la scheda principale è la prima scheda visualizzata durante l'esecuzione dell'applicazione.

Per cambiare la scheda principale del progetto:

1 Scegliere Project | Options e selezionare la pagina Forms.

2 Nella casella combinata Main Form selezionare la scheda che si vuole utilizzare come scheda principale del progetto e scegliere OK.

A questo punto, se si esegue l'applicazione, verrà visualizzata la scheda selezionata come scheda principale.

Occultamento della scheda principale

In fase di avvio dell'applicazione, è possibile fare in modo che la scheda principale non venga visualizzata. Per fare ciò, occorre utilizzare la variabile globale *Application* (descritta nel paragrafo , "Il lavoro a livello di applicazione", a pagina 8-2).

Per nascondere la scheda principale in fase di avvio:

1 Scegliere l'opzione Project | View Source per visualizzare il file principale del progetto.

2 Aggiungere le seguenti righe di codice subito dopo la chiamata a `Application->CreateForm()` e prima della chiamata a `Application->Run()`.

```
Application->ShowMainForm = false;
```

3 Usando l'Object Inspector, impostare la proprietà *Visible* della scheda principale a **false**.

Aggiunta di schede

Per aggiungere una scheda al progetto, occorre selezionare *File* | New | Form. È possibile vedere tutte le schede del progetto e le loro unit associate elencate nel Project Manager (View | Project Manager) e, scegliendo il comando View | Forms, è possibile visualizzare un elenco delle schede non associate.

Collegamento delle schede

Aggiungendo una scheda a un progetto, si aggiunge nel relativo file un riferimento al progetto, ma non alle altre unit del progetto. Prima di scrivere il codice che fa riferimento alla nuova scheda, occorre aggiungere un riferimento alla scheda stessa

nei file unit delle schede di referenziazione. Questa operazione viene detta *collegamento della scheda*.

Uno dei motivi più comuni per collegare le schede è quello di fornire l'accesso ai relativi componenti. Per esempio, questo collegamento verrà spesso usato per consentire a una scheda che contiene componenti associati ai dati di collegarsi ai componenti di accesso ai dati di un modulo dati.

Per collegare una scheda a un'altra:

1 Selezionare la scheda che ha bisogno di fare riferimento a un'altra scheda.
2 Scegliere File | Include Unit Hdr.
3 Selezionare il nome della unit della scheda a cui bisogna fare riferimento.
4 Scegliere OK.

Collegare una scheda a un'altra vuole dire che la unit della scheda contiene l' intestazione per la unit della scheda dell'altra, ovvero che la scheda collegata e i suoi componenti sono nel campo d'azione della scheda collegante.

Gestione del layout

Nel caso più semplice il layout dell'interfaccia utente viene controllato dal modo in cui i controlli vengono collocati nelle schede. Le scelte di collocazione effettuate vengono riflesse nelle proprietà *Top*, *Left*, *Width*, e *Height* del controllo. È possibile cambiare questi valori in fase di esecuzione per cambiare la posizione e le dimensioni dei controlli nelle schede.

I controlli hanno, comunque, una grande quantità di altre proprietà che consentono loro di regolare automaticamente il loro contenuto o i loro contenitori. Ciò consente di preparare le schede in modo che le diverse parti costituiscano un tutt'uno.

Due proprietà incidono sul modo in cui un controllo viene posizionato e dimensionato rispetto al suo genitore. La proprietà *Align* permette di forzare un controllo per sistemarlo perfettamente all'interno del suo genitore lungo un determinato margine o riempiendo l'intera area client dopo uno dei controlli che sono stati allineati. Quando il genitore viene ridimensionato, i controlli ad esso allineati vengono ridimensionati automaticamente e rimangono posizionati in modo da adattarsi rispetto a un determinato margine.

Se si vuole tenere un controllo posizionato su un particolare margine del suo genitore, senza che però venga ridimensionato per essere sistemato lungo l'intero margine, basta usare la proprietà *Anchors*.

Se si vuole avere la certezza che un controllo non diventi troppo grande o troppo piccolo, si può ricorrere alla proprietà *Constraints*. *Constraints* permette di specificare l'altezza massima e minima, e la larghezza massima e minima del controllo. Impostando questi elementi, si limitano le dimensioni (in pixel) dell'altezza e della larghezza del controllo. Per esempio, impostando *MinWidth* e *MinHeight* dei vincoli su un oggetto contenitore, si può garantire che gli oggetti figlio siano sempre visibili.

Il valore di *Constraints* si propaga attraverso la gerarchia genitore/figlio, in modo che le dimensioni di un oggetto possano essere vincolate, dal momento che esso contiene oggetti figlio allineati soggetti al vincolo delle dimensioni. *Constraints* può anche

impedire che un controllo venga ridotto rispetto a una certa dimensione quando viene chiamato il suo metodo *ChangeScale*.

TControl introduce un evento protected, *OnConstrainedResize*, di tipo *TConstrainedResizeEvent*:

```
void __fastcall (__closure *TConstrainedResizeEvent)(System::TObject* Sender, int
&MinWidth, int &MinHeight, int &MaxWidth, int &MaxHeight);
```

Questo evento consente di ridefinire i vincoli delle dimensioni quando si cerca di ridimensionare il controllo. I valori dei vincoli vengono passati come parametri var che possono essere modificati all'interno del gestore di evento. I valori dei vincoli vengono passati come parametri varche possono essere modificati all'interno del gestore di evento. *OnConstrainedResize* è published per gli oggetti contenitore (*TForm*, *TScrollBox*, *TControlBar* e *TPanel*). Inoltre, gli scrittori di componenti possono usare o pubblicare questo evento per qualsiasi discendente di *TControl*.

I controlli con un contenuto in grado di cambiare le dimensioni hanno una proprietà *AutoSize*, che obbliga il controllo a regolare le dimensioni in funzione degli oggetti contenuti.

Uso delle schede

Quando si crea una scheda dall'IDE, C++Builder crea automaticamente la scheda in memoria includendo il codice nel punto di ingresso principale della funzione dell'applicazione. Solitamente, questo è il comportamento desiderato e non occorre fare nulla per modificarlo. La finestra principale rimane, cioè, sullo schermo per tutta la durata del programma, e quindi probabilmente non sarà necessario modificare il comportamento predefinito di C++Builder quando si crea la scheda per la finestra principale.

Tuttavia, può risultare inopportuno avere in memoria tutte le schede dell'applicazione per la durata dell'esecuzione del programma. Se, cioè, si preferisce che le finestre di dialogo dell'applicazione non stiano tutte in memoria contemporaneamente, è possibile creare dinamicamente le finestre di dialogo quando si vuole che vengano visualizzate.

Le schede possono essere modali o non modali. Le schede modali sono schede con le quali l'utente deve interagire prima di passare a un'altra scheda (per esempio, una finestra di dialogo che richiede l'immissione dell'utente). Le schede non modali, invece, sono finestre che vengono visualizzate finché non vengono coperte da un'altra finestra o non vengono chiuse o ridotte a icona dall'utente.

Controllo delle schede in memoria

Per impostazione predefinita, C++Builder crea automaticamente in memoria la scheda principale dell'applicazione includendo il seguente codice nel punto d'ingresso principale dell'applicazione:

```
Application ->CreateForm(__classid(TForm1), &Form1);
```

Questa funzione crea una variabile globale con lo stesso nome della scheda. Quindi, ogni scheda contenuta in un'applicazione ha una variabile globale associata. Questa variabile è un puntatore a un'istanza della classe della scheda e viene usata per fare riferimento alla scheda mentre l'applicazione è in esecuzione. Ogni file di codice sorgente (.cpp) che include il file header (.h) della scheda può accedere alla scheda tramite questa variabile.

Poiché la scheda viene aggiunta nel punto d'ingresso principale dell'applicazione, la scheda appare non appena si avvia il programma e resta in memoria per tutta la durata dell'applicazione.

Visualizzazione di una scheda creata automaticamente

Se si sceglie di creare una scheda in fase di avvio, che si vuole visualizzare solo più tardi durante l'esecuzione del programma, il gestore di evento della scheda usa il metodo *ShowModal* per visualizzare la scheda già caricata memoria:

```
void __fastcall TMainMForm::FirstButtonClick(TObject *Sender)
{
   ResultsForm->ShowModal();
}
```

In questo caso, poiché la scheda è già in memoria, non c'è alcuna necessità di crearne un'altra istanza o di distruggerla.

Creazione dinamica delle schede

È probabile che non si voglia che tutte le schede dell'applicazione stiano in memoria contemporaneamente. Per ridurre la quantità di memoria necessaria al momento del caricamento, può essere opportuno creare alcune schede solo quando c'è bisogno di usarle. Per esempio, una finestra di dialogo ha bisogno di trovarsi in memoria solo durante il periodo in cui un utente la utilizza.

Per creare una scheda in una fase diversa durante l'esecuzione usando l'IDE, occorre:

1. Selezionare File|New Form dal menu principale per visualizzare la nuova scheda.
2. Rimuovere la scheda dall'elenco Auto-create forms della pagina Project|Options|Forms.

   Questa operazione rimuove la chiamata alla scheda. In alternativa, si può rimuovere manualmente la seguente riga dal punto d'ingresso principale del programma:

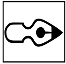
   ```
   Application->CreateForm(__classid(TResultsForm), &ResultsForm);
   ```
3. Chiamare la scheda nel momento desiderato, usando il metodo *Show* della scheda, se essa è non modale, o il metodo *ShowModal*, se è modale.

Un gestore di evento per la scheda principale deve creare un'istanza della scheda risultato e distruggerla. Un modo per chiamare la scheda risultato consiste nell'usare la variabile globale nel modo seguente. Si noti che *ResultsForm* è una scheda modale, quindi il gestore usa il metodo *ShowModal*.

```
void __fastcall TMainMForm::FirstButtonClick(TObject *Sender)
{
```

```
    ResultsForm = new TResultsForm(this);
  ResultsForm->ShowModal();
  delete ResultsForm;
}
```

Il gestore di evento dell'esempio cancella la scheda dopo averla chiusa, in modo che non occorrerà ricrearla se occorrerà usare *ResultsForm* in qualche altro punto dell'applicazione. Se la scheda fosse visualizzata usando *Show*, non potrebbe essere cancellata all'interno del gestore di evento, perché *Show* termina mentre la scheda è ancora aperta.



Se si crea una scheda usando l'operatore **new**, bisogna ricordarsi di controllare che non sia nell'elenco Auto-create forms della pagina Project Options|Forms. Nello specifico, se la nuova scheda viene creata senza cancellare la scheda omonima dall'elenco, C++Builder crea la scheda in fase di avvio e questo gestore di evento crea una nuova istanza della scheda, sovrascrivendo il riferimento all'istanza creata automaticamente. Questa istanza esiste ancora, ma l'applicazione non può più accedervi. Dopo che il gestore di evento ha terminato l'esecuzione, la variabile globale non punta più a una scheda valida. Qualsiasi tentativo di dereferenziare la variabile globale bloccherà probabilmente l'applicazione.

Creazione di schede non modali come finestre

Occorre garantire che le variabili di riferimento per le schede non modali esistano finché la scheda è in uso. Questo vuol dire che queste variabili devono avere un campo d'azione globale. Nella maggior parte dei casi si usa la variabile globale di riferimento creata insieme alla scheda (il nome della variabile che corrisponde alla proprietà name della scheda). Se l' applicazione richiede altre istanze della scheda, occorre dichiarare variabili globali distinte (di tipo puntatore alla classe form) una per ciascuna istanza.

Creazione di un'istanza di scheda mediante una variabile locale

Un modo più sicuro per creare un'istanza univoca di una *scheda modale* consiste nell'usare una variabile locale nel gestore di evento come riferimento a una nuova istanza. Se si usa una variabile locale, non importa se *ResultsForm* è stata creata automaticamente o meno. Il codice nel gestore di evento non fa alcun riferimento alla variabile globale della scheda. Per esempio:

```
void __fastcall TMainMForm::FirstButtonClick(TObject *Sender)
{
    TResultsForm *rf = new TResultsForm(this);// rf is local form instance
  rf->ShowModal();
  delete rf;      // form safely destroyed
}
```

Si noti che l'istanza globale della scheda non viene mai usata in questa versione del gestore di evento.

Normalmente, le applicazioni usano le istanze globali delle schede. Tuttavia, se occorre una nuova istanza di una scheda modale, e si usa quella scheda in una sezione limitata dell'applicazione, come una singola funzione, un'istanza locale è solitamente il modo più sicuro ed efficace per lavorare con la scheda.

Naturalmente, non è possibile usare le variabili locali nei gestori di evento per le schede non modali, perché devono avere un campo d'azione globale per garantire che le schede esistano finché sono utilizzate. *Show* termina non appena si apre la scheda, quindi se è stata usata una variabile locale, questa variabile uscirà immediatamente dal campo d'azione.

Passaggio di altri argomenti alle schede

Normalmente, le schede per l'applicazione vengono create dall'interno dell'IDE. Quando vengono create in questo modo, le schede hanno un costruttore che accetta un argomento, *Owner*, che è il proprietario della scheda da creare. (Il proprietario è l'oggetto applicazione o scheda chiamante). *Owner* può essere NULL.

Per passare alla scheda altri argomenti, si deve creare un altro costruttore e istanziare la scheda usando l' operatore **new**. La classe della scheda di esempio sotto riportata mostra un costruttore aggiuntivo, con un ulteriore argomento *whichButton*. Questo nuovo costruttore viene aggiunto manualmente alla classe della scheda.

```
class TResultsForm : public TForm
{
__published:    // IDE-managed Components
   TLabel *ResultsLabel;
   TButton *OKButton;
   void __fastcall OKButtonClick(TObject *Sender);
private:        // User declarations
public:         // User declarations
   virtual __fastcall TResultsForm(TComponent* Owner);
   virtual __fastcall TResultsForm(int whichButton, TComponent* Owner);

};
```

Ecco il costruttore codificato manualmente che passa l'ulteriore argomento, *whichButton*. Questo costruttore usa il parametro *whichButton* per impostare la proprietà *Caption* di un controllo *Label* sulla scheda.

```
void__fastcall TResultsForm::TResultsForm(int whichButton, TComponent* Owner)
 : TForm(Owner)
{
   switch (whichButton) {
      case 1:
         ResultsLabel->Caption = "You picked the first button!";
         break;
      case 2:
         ResultsLabel->Caption = "You picked the second button!";
         break;
      case 3:
         ResultsLabel->Caption = "You picked the third button!";
   }
}
```

Quando si crea un'istanza di una scheda con più costruttori è possibile selezionare il costruttore più adatto allo scopo. Per esempio, il seguente gestore *OnClick* per un pulsante su una scheda crea un'istanza di *TResultsForm*, che usa il parametro ulteriore:

```
void __fastcall TMainMForm::SecondButtonClick(TObject *Sender)
{
   TResultsForm *rf = new TResultsForm(2, this);
   rf->ShowModal();
   delete rf;
}
```

Recupero dei dati dalle schede

La maggior parte delle applicazioni vere e proprie è costituita da più schede. Spesso, occorre passare le informazioni tra queste schede. Le informazioni possono essere passate a una scheda per mezzo di parametri al costruttore della scheda ricevente, o assegnando i valori alle proprietà della scheda. Il modo in cui si ottengono informazioni da una scheda dipende dal fatto che la scheda sia modale o non modale.

Recupero dei dati da schede non modali

È possibile estrarre con facilità informazioni dalle schede non modali chiamando funzioni membro public della scheda o sottoponendo a query le proprietà della scheda. Per esempio, si supponga che un’applicazione contenga una scheda non modale di nome *ColorForm*, che contiene una casella di riepilogo chiamata *ColorListBox* con un elenco di colori (“Red”, “Green”, “Blue”, e così via). La stringa del nome del colore selezionato in *ColorListBox* viene memorizzata automaticamente in una proprietà di nome *CurrentColor* ogni volta che un utente seleziona un nuovo colore. La dichiarazione di classe per la scheda è la seguente:

```
class TColorForm : public TForm
{
   __published:    // IDE-managed Components
      TListBox *ColorListBox;
      void __fastcall ColorListBoxClick(TObject *Sender);
   private:        // User declarations
      String getColor();
      void   setColor(String);
      String curColor;
   public:         // User declarations
      virtual __fastcall TColorForm(TComponent* Owner);
      __property String CurrentColor = {read=getColor, write=setColor};
};
```

Il gestore di evento *OnClick* per la casella di riepilogo, *ColorListBoxClick*, imposta il valore della proprietà *CurrentColor* ogni volta che viene selezionato un nuovo elemento nella casella di riepilogo. Il gestore di evento acquisisce la stringa dalla casella di riepilogo contenente il nome del colore e la assegna a *CurrentColor*. La proprietà *CurrentColor* usa la funzione setter, *setColor*, per memorizzare il valore effettivo per la proprietà nel membro dati private *curColor*:

```
void __fastcall TColorForm::ColorListBoxClick(TObject *Sender)
{
   int index = ColorListBox->ItemIndex;
   if (index >= 0) {// make sure a color is selected
      CurrentColor = ColorListBox->Items->Strings[index];
   }
```

```
   else          // no color selected
      CurrentColor = "";
}
//----------------------------------------------------------------------
void TColorForm::setColor(String s)
{
   curColor = s;
}
```

A questo punto, si supponga che un'altra scheda all'interno dell'applicazione, chiamata *ResultsForm*, debba scoprire quale colore è al momento selezionato su *ColorForm* ogni volta che si fa clic su un pulsante (chiamato *UpdateButton)* di *ResultsForm***.** Il gestore di evento *OnClick* per *UpdateButton* potrebbe avere il seguente formato:

```
void __fastcall TResultsForm::UpdateButtonClick(TObject *Sender)
{
   if (ColorForm) {// verify ColorForm exists
      String s = ColorForm->CurrentColor;
      // do something with the color name string
   }
}
```

Per prima cosa il gestore di evento verifica che *ColorForm* esista controllando se il puntatore è NULL. Quindi, riceve il valore della proprietà *CurrentColor* di *ColorForm***.** La richiesta del *CurrentColor* chiama la sua funzione getter *getColor* mostrata di seguito:

```
String TColorForm::getColor()
{
   return curColor;
}
```

In caso contrario, se la funzione *getColor* di *ColorForm* fosse pubblica, un'altra scheda potrebbe acquisire il colore attivo senza usare la proprietà *CurrentColor* (per esempio, String *s* = *ColorForm->getColor*();). Infatti, non esiste un modo per impedire che un'altra scheda riceva il colore di *ColorForm* al momento selezionato contrassegnando direttamente le selezione della casella di riepilogo:

```
String s = ColorListBox->Items->Strings[ColorListBox->ItemIndex];
```

Tuttavia, l'uso di una proprietà rende l'interfaccia per *ColorForm* molto chiara e semplice. Tutto ciò che una scheda ha bisogno di sapere su *ColorForm* è come controllare il valore di *CurrentColor*.

Recupero dei dati da schede modali

A differenza delle schede non modali, spesso le schede modali contengono le informazioni necessarie per altre schede. L'esempio più comune è una scheda A che lancia una scheda modale B. Quando la scheda B viene chiusa, la scheda A ha bisogno di conoscere che cosa ha fatto l'utente con la scheda B per decidere come procedere con l'elaborazione della scheda A. Se la scheda B è ancora in memoria, può essere sottoposta a query tramite proprietà o funzioni membro esattamente come nell'esempio sulle schede non modali sopra riportato. Ma come gestire le situazioni in cui la scheda B viene cancellata dalla memoria alla chiusura? Poiché una scheda

non ha una valore di ritorno esplicito, occorre preservare le informazioni importanti della scheda prima che venga distrutta.

Per illustrare ciò, si prenderà ora in considerazione una versione modificata della scheda *ColorForm* che è progettata per essere una scheda modale. La dichiarazione di classe è la seguente:

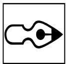
```
class TColorForm : public TForm
{
   __published:    // IDE-managed Components
      TListBox *ColorListBox;
      TButton *SelectButton;
      TButton *CancelButton;
      void __fastcall CancelButtonClick(TObject *Sender);
      void __fastcall SelectButtonClick(TObject *Sender);
   private:        // User declarations
      String* curColor;
   public:         // User declarations
      virtual __fastcall TColorForm(TComponent* Owner);
      virtual __fastcall TColorForm(String* s, TComponent* Owner);
};
```

La scheda ha una casella di riepilogo chiamata *ColorListBox* con un elenco di nomi di colori. Quando viene premuto, il pulsante di nome *SelectButton* prende nota del nome del colore attualmente selezionato in *ColorListBox* e chiude la scheda. *CancelButton* è un pulsante che chiude semplicemente la scheda.

Si noti che è stato aggiunto un costruttore definito dall'utente alla classe che accetta un argomento *String**. Presumibilmente, questo *String** punta a una stringa che indica la scheda che lancia *ColorForm*. L'implementazione di questo costruttore è la seguente:

```
void__fastcall TColorForm::TColorForm(String* s, TComponent* Owner)
 : TForm(Owner)
{
   curColor = s;
   *curColor = "";
}
```

Il costruttore salva il puntatore in un membro dati privato *curColor* e inizializza la stringa a una stringa vuota.



Per usare il costruttore definito dall'utente sopra riportato, la scheda deve essere creata esplicitamente. Essa non può essere creata automaticamente quando l'applicazione è avviata. Per informazioni dettagliate, consultare il paragrafo "Controllo delle schede in memoria" a pagina 8-5.

Nell'applicazione, l'utente seleziona un colore dalla casella di riepilogo e preme *SelectButton* per salvare la scelta e chiudere la scheda. Il gestore di evento *OnClick* per *SelectButton* potrebbe avere questa forma:

```
void __fastcall TColorForm::SelectButtonClick(TObject *Sender)
{
   int index = ColorListBox->ItemIndex;
   if (index >= 0)
      *curColor = ColorListBox->Items->Strings[index];
```

```
      Close();
   }
```

Notare che il gestore di evento memorizza il nome del colore selezionato nella stringa address che è stata passata al costruttore.

Per usare efficacemente *ColorForm*, la scheda chiamante deve passare il costruttore a un puntatore a una stringa esistente. Per esempio, si supponga che *ColorForm* è stata istanziata da una scheda chiamata *ResultsForm* in risposta a un clic su un pulsante chiamato *UpdateButton* di *ResultsForm*. Il gestore di evento dovrebbe essere simile al seguente codice:

```
void __fastcall TResultsForm::UpdateButtonClick(TObject *Sender)
{
   String s;
   GetColor(&s);
   if (s != "") {
      // do something with the color name string
   }
   else {
      // do something else because no color was picked
   }
}
//---------------------------------------------------------------------
void TResultsForm::GetColor(String *s)
{
   ColorForm = new TColorForm(s, this);
   ColorForm->ShowModal();
   delete ColorForm;
   ColorForm = 0; // NULL the pointer
}
```

UpdateButtonClick crea una stringa chiamata *s*. L'indirizzo di *s* viene passato alla funzione *GetColor* che crea *ColorForm*, passando il puntatore a *s* come argomento per il costruttore. *ColorForm* viene cancellata non appena viene chiusa, ma il nome del colore che è stato selezionato è ancora protetto in *s*, presupponendo che è stato selezionato un colore. Altrimenti, *s* contiene una stringa vuota che una chiara indicazione che l'utente è uscito da *ColorForm* senza selezionare un colore.

Questo esempio usa una variabile string per contenere informazioni dalla scheda modale. Naturalmente, possono essere usati oggetti più complessi a seconda delle necessità. Tenere ben presente che occorre sempre fornire un modo per consentire alla scheda chiamante di sapere se la scheda modale è stata chiusa senza apportare modifiche o effettuare selezioni (con *s* impostato, per impostazione predefinita, a una stringa vuota).

Riutilizzo di componenti e gruppi di componenti

C++Builder offre molti modi per salvare e riutilizzare il lavoro fatto con i componenti:

- L'uso di *modelli di componenti* rappresenta un modo semplice e veloce per configurare e salvare gruppi di componenti. Consultare il paragrafo "Creazione e utilizzo di modelli di componenti" a pagina 8-13.
- Nel Repository è possibile salvare *schede*, *moduli dati* e *progetti*. In questo modo si dispone di un database centrale di elementi riutilizzabili ed è possibile usare l'ereditarietà delle schede per propagare le modifiche.
- È possibile salvare i *frame* nel Repository o nella Component palette. I frame utilizzano l'ereditarietà delle schede e possono essere incorporati nelle schede o in altri frame. Consultare il paragrafo "Operazioni con i frame" a pagina 8-14.
- La creazione di *componenti custom* è il sistema più complicato per riutilizzare il codice, ma offre una maggiore flessibilità. Consultare il Capitolo 45, "Introduzione alla creazione di componenti".

Creazione e utilizzo di modelli di componenti

È possibile creare modelli costituiti da uno o più componenti. Dopo aver disposto dei componenti su una scheda, impostato le rispettive proprietà e aver scritto il codice relativo, li si può salvare come un *modello di componente*. In seguito, selezionando il modello dalla Component palette, è possibile collocare su una scheda i componenti prestabiliti con una sola operazione; tutte le proprietà associate e il codice di gestione degli eventi vengono aggiunti contemporaneamente al progetto.

Una volta collocato un modello su una scheda, è possibile riposizionare i componenti in modo indipendente, azzerare le proprietà, e creare o modificare i gestori di evento esattamente come se si fosse collocato ciascun componente con un'operazione separata.

Per creare un modello di componente:

1. Collocare e disporre dei componenti su una scheda. Nell'Object Inspector, impostarne le proprietà e gli eventi come si desidera.
2. Selezionare i componenti. Il modo più semplice per selezionare molti componenti consiste nel trascinare il mouse su tutti i componenti. Agli vertici di ciascun componente selezionato appariranno delle maniglie di color grigio.
3. Scegliere il comando Component|Create Component Template.
4. Specificare nella casella di testo Component Template Information un nome da assegnare al modello di componente. La proposta predefinita è il tipo componente del primo componente selezionato al punto 2, seguito dalla parola "Template". Ad esempio, se si seleziona un'etichetta e quindi una casella di modifica, il nome proposto sarà "TLabelTemplate". È possibile modificare questo nome, ma fare attenzione a non duplicare nomi di componenti esistenti.
5. Nella casella di testo Palette page, specificare la pagina della Component palette in cui si desidera collocare il modello. Se si specifica una pagina inesistente, nel momento in cui si salverà il modello verrà creata una nuova pagina.
6. Sotto Palette Icon, selezionare una bitmap che servirà a rappresentare il modello sulla tavolozza. La bitmap proposta per impostazione predefinita sarà la bitmap utilizzata dal tipo di componente del primo componente selezionato al punto 2.

Per cercare altre bitmap, fare clic su Change. La bitmap selezionata non deve essere più grande di 24 pixel per 24 pixel.

7 Fare click su OK.

Per rimuovere i modelli dalla Component palette, scegliere il comando Component | Configure Palette.

Operazioni con i frame

Un frame (*TFrame*), analogamente a una scheda, è un contenitore per altri componenti. Esso utilizza lo stesso meccanismo di possesso delle schede per l'istanzazione e la distruzione dei componenti in esso presenti, e le stessa relazioni genitore-figlio per la sincronizzazione delle proprietà dei componenti.

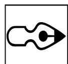
Per certi versi, un frame è più simile a un componente custom che a una scheda. Una volta creato e salvato un frame, esso continua a funzionare come una unit, ereditando le modifiche dai componenti (inclusi altri frame) in esso contenuti. Una volta creato e salvato un frame, esso continua a funzionare come una unit, ereditando le modifiche dai componenti (inclusi altri frame) in esso contenuti. Quando si include un frame in un'altro frame o in una scheda, esso continua ad ereditare le modifiche apportate al frame da cui è stato derivato.

I frame sono utili per organizzare gruppi di controlli che vengono utilizzati in più punti dell'applicazione. Ad esempio, se si ha una bitmap che viene utilizzata su più schede, è possibile collocarla in un frame e solo una copia di quella bitmap verrà inclusa nelle risorse dell'applicazione. Si potrebbe anche includere in un frame una serie di campi di editing, il cui scopo è quello di consentire l'editing di una tabella, e utilizzare quel frame ogni volta che si desidera inserire dati nella tabella.

Creazione di frame

Per creare un frame vuoto, scegliere il comando *File* | New | Frame oppure scegliere *File* | New | Other e fare doppio clic su Frame. A questo punto è possibile trascinare sul nuovo frame altri componenti (inclusi altri frame.

Di solito è buona norma–anche se non strettamente necessario–salvare i frame come parte di un progetto. Se si vuole creare un progetto che contenga solo frame e nessuna scheda, scegliere il comando *File* | New | *Application*, chiudere la nuova scheda e la unit senza salvarle, quindi scegliere il comando *File* | New | Frame e salvare il progetto.



Quando si salvano i frame, è bene evitare l'uso dei nomi predefiniti, quali *Unit1*, *Project1* e così via, in quanto molto probabilmente provocheranno conflitti quando si cercherà di utilizzare il frame in un secondo tempo.

In fase di progetto, è possibile visualizzare tutti i frame inclusi nel progetto corrente scegliendo il comando View | Forms e selezionando un frame. Come con le schede e i moduli dati, è possibile passare dalla visualizzazione grafica nel Form Designer al

file .form del frame, e viceversa, facendo clic destro e scegliendo il comando View as Form oppure View as Text.

Aggiunta di frame alla Component palette

I frame vengono aggiunti alla Component palette come modelli di componenti. Per aggiungere un frame alla Component palette, aprire il frame nel Form Designer (a questo scopo non è possibile utilizzare un frame incluso in un altro componente), fare clic destro sul frame e scegliere il comando Add della Palette. Nella finestra di dialogo Component Template Information che si aprirà, selezionare un nome, la pagina della tavolozza e l'icona da assegnare al nuovo modello.

Uso e modifica di frame

Per utilizzare un frame in un'applicazione, è necessario collocarlo, direttamente o indirettamente, su una scheda. È possibile aggiungere frame direttamente alle schede, ad altri frame o ad altri oggetti contenitore come i pannelli e le caselle a scorrimento.

Form Designer consente di aggiungere un frame a un'applicazione in due modi:

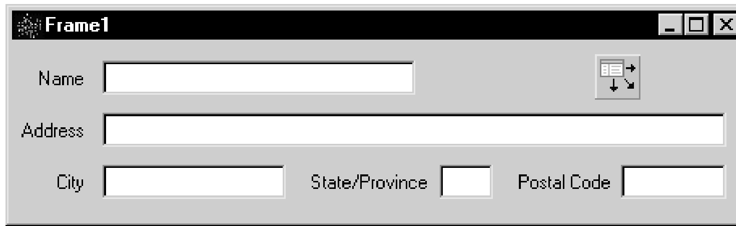
- Selezionare un frame nella Component palette e trascinarlo su una scheda, su un altro frame o su un altro oggetto contenitore. Se occorre, Form Designer chiede il permesso di includere il file unit del frame nel progetto.
- Selezionare *Frames* dalla pagina Standard della Component palette e fare clic su una scheda o su un altro frame. Apparirà una finestra di dialogo con l'elenco dei frame che sono già stati inclusi nel progetto; selezioanrne uno e fare clic su OK.

Quando si trascina un frame su una scheda o su un altro contenitore, C++Builder dichiara una nuova classe che deriva dal frame selezionato. (In modo analogo, quando si aggiunge una nuova scheda a un progetto, C++Builder dichiara una nuova classe che deriva da *TForm*.) Ciò significa che le modifiche fatte successivamente al frame originale (l'antenato) si propagano al frame incorporato, mentre le modifiche apportate al frame incorporato non si propagano a ritroso all'antenato.

Si supponga, ad esempio, di voler assemblare un gruppo di componenti per l'acecsso ai dati e di controlli data-aware da utilizzare più volte, anche in più di una applicazione. Uno dei modi per fare ciò potrebbe essere quello di raccogliere i componenti in un modello di componenti; ma, se si iniziasse ad usare il modello e poi si cambiasse idea rispetto alla disposizione dei controlli, si dovrebbe tornare indietro e modificare manualmente tutti i progetti in cui è stato collocato il modello.

Invece, se si collocassero i componenti database in un frame, le eventuali modifiche dovrebbero essere apportate in un solo posto; le modifiche al frame originale si propagano ai discendenti in esso contenuti al momento della ricompilazione del progetto. Allo stesso tempo, si è liberi di modificare qualunque frame incluso senza toccare il frame originale o altri suoi discendenti incorporati. L'unica limitazione nella modifica di frame incorporati consiste nel non poter aggiungere ad essi altri componenti.

Figura 8.1 Un frame con controlli data-aware e un componente datasource



Oltre a sempliifcare la manutenzione, i frame risultano utili nell'utilizzare le risorse in modo più efficiente. Ad esempio, per utilizzare in una applicazione una bitmap o un'altra immagine grafica, si dovrebbe caricare l'immagine grafica nella proprietà *Picture* di un controllo *TImage*. Se nell'applicazione la stessa immagine viene utilizzata più volte, ciuscun oggetto *Image* collocato su una scheda provocherà l'aggiunta al file di risorse della scheda di un'altra copia dell'immagine. (Questo è vero anche se si imposta *TImage::Picture* una volta per tutte e si salva il controllo *Image* come modello di componente). La soluzione migliore consiste nel trascinare un oggetto *Image* su un frame, caricarvi l'immagine grafica e quindi usare il frame nel punto in cui si vuole far apparire l'immagine. In questo modo i file scheda prodotti risultano più piccoli oltre ad offrire la possibilità di cambiare globalmente l'immagine utilizzata, semplicemente modificando la proprietà *Image* sul frame originale.

Condivisione di frame

È possibile condividere un frame con altri sviluppatori in due modi:

- Aggiungendo il frame all' Object Repository.
- Distribuendo i file unit (.cpp e h.) e i file scheda (.dfm o .xfm).

Per aggiungere un frame al Repository, aprire un qualsiasi progetto contenente il frame, fare clic destro in Form Designer e scegliere il comando Add to Repository. Per maggiori informazioni, consultare "Utilizzo dell'Object Repository" a pagina 7-25.

Se si distribuiscono i file unit e scheda ad altri sviluppatori, essi potranno aprirli e aggiungerli alla Component palette. Se il frame incorpora altri frame dovranno aprire il frame come parte di un progetto.

Sviluppo di finestre di dialogo

I componenti finestra di dialogo della pagina Dialogs della Component palette rendono disponibili alle applicazioni diverse finestre di dialogo. Queste finestre di dialogo forniscono alle applicazioni un'interfaccia familiare e uniforme e permettono all'utente di eseguire le comuni operazioni sui file, come l'apertura e il salvataggio di file. Le finestre di dialogo servono per visualizzare e/o ottenere dati.

Ogni finestra di dialogo si apre quando viene chiamato il metodo *Execute*. *Execute* restituisce un valore Boolean: se l'utente sceglie OK per accettare le modifiche apportate nella finestra di dialogo, *Execute* restituisce **true**; se l'utente sceglie Cancel per uscire dalla finestra di dialogo senza fare modifiche oppure senza salvarle, *Execute* restituisce **false**.

Se si stanno sviluppando applicazioni multipiattaforma, è possibile utilizzare le finestre di dialogo fornite con CLX nella unit QDialogs. Per i sistemi operativi che hanno tipi di finestra di dialogo native per le comuni attività, come l'apertura o il salvataggio dei file oppure per la modifica di font o colore, è possibile utilizzare la proprietà *UseNativeDialog* . Impostare *UseNativeDialog* a **true** se l'applicazione sarà eseguirà in tale ambiente e se si desidera che utilizzi le finestre di dialogo native invece delle finestre di dialogo Qt.

Utilizzo delle finestre di dialogo di tipo Open

Uno dei componenti finestra di dialogo più comunemente utilizzato è *TOpenDialog*. Questo componente di solito viene richiamato da una voce di menu New o Open inclusa nel menu *File* della barra del menu principale di una scheda. La finestra di dialogo contiene diversi controlli che consentono di selezionare gruppi di file utilizzando i carattere jolly e di spostarsi nella struttura delle directory.

Il componente *TOpenDialog* rende disponibile all'applicazione una finestra di dialogo Open. Lo scopo di questa finestra di dialogo è quello di consentire che un utente selezioni un file da aprire. Per visualizzare la finestra di dialogo si utilizza il metodo *Execute*.

Se l'utente sceglie OK nella finestra di dialogo, il file selezionato dall'utente viene memorizzato nella proprietà FileName di *TOpenDialog*, e lo si potrà elaborare quindi a proprio piacimento.

Il frammento di codice seguente può essere collocato in un *Action* e collegato alla proprietà *Action* di un sottoelemento di *TMainMenu* oppure collocato nell'evento *OnClick* del sottoelemento:

```
if(OpenDialog1->Execute()){
   filename = OpenDialog1->FileName;
};
```

Questo codice visualizzerà la finestra di dialogo e se l'utente preme il pulsante OK, copierà il nome del file in una variabile di tipo *AnsiString*, dichiarata in precedenza, di nome `filename`.

Organizzazione delle azioni per barre e menu

C++Builder dispone di numerose funzioni che semplificano la creazione, la personalizzazione e la gestione dei menu e delle barre strumenti. Queste funzioni permettono di organizzare liste di azioni che possono essere avviate dagli utenti dell'applicazione mediante la pressione di un pulsante su una barra strumenti,

scegliendo un comando di un menu oppure puntando un'icona e facendo clic su di essa.

Spesso un insieme di azioni viene utilizzato da più elementi dell'interfaccia utente. Ad esempio, i comandi Cut, Copy e Paste spesso vengono visualizzati sia in un menu Edit sia in una barra strumenti. Per utilizzare l'azione in più elementi dell'interfaccia utente dell'applicazione è sufficiente aggiungere l'azione una sola volta.

Per la piattaforma Windows, sono disponibili una serie di strumenti per semplificare la definizione e il raggruppamento di azioni, la creazione di layout differenti e la personalizzazone di menu in progettazione o in esecuzione. Questi strumenti sono noti con il nome collettivo di ActionBand, e i menu e le barre strumenti che con essi si creano vengono definiti bande di azioni. Di solito, è possibile creare un'interfaccia utente ActionBand come segue:

- Creare la lista di azioni per creare un insieme di azioni che sarà disponibile per l'applicazione (utilizzare l'Action Manager, *TActionManager*)
- Aggiungere all'applicazione gli elementi dell'interfaccia utente (utilizzare componenti ActionBand come *TActionMainMenuBar* e *TActionToolBar*)
- Trascinare e rilasciare le azioni dall'Action Manager sugli elementi dell'interfaccia utente

La tabella seguente definisce la terminologia relativa alla configurazione dei menu e delle barre strumenti:

Tabella 8.1 Terminologia per la configurazione delle azioni

Termine	Definizione
Azione	Una risposta a qualcosa fatto dall'utente, come il clic su una voce di menu. Molte azioni standard, necessarie con una certa frequenza, sono già disponibili e utilizzabili nelle applicazioni così come sono. Ad esempio, le operazioni sui file, come File Open, File SaveAs, File Run e File Exit, sono già incluse, insieme a molte altre per l'editing, la formattazione, la ricerca, l'attivazione dell'help, per le finestre di dialogo e per le azioni di finestra. Utilizzando liste di azioni e l'*Action* Manager è possibile anche programmare azioni custom e accedervi.
Banda di azioni	Un contenitore per un insieme di azioni associate a un menu o a una barra strumenti personalizzabile. I componenti ActionBand per i menu principali e per le barre strumenti (*TActionMainMenuBar* e *TActionToolBar*) sono esempi di bande di azioni.
Categoria di azioni	Consente di raggruppare azioni e rilasciarle come gruppo su un menu o su una barra strumenti. Ad esempio, una delle categorie di azione standard è Search che include le azioni Find, FindFirst, FindNext e Replace.
Classi di azioni	Classi che eseguono le azioni utilizzate nell'applicazione. Tutte le azioni standard sono definite in classi di azioni come *TEditCopy*, *TEditCut* e *TEditUndo*. È possibile utilizzare queste classi trascinandole dalla finestra di dialogo Customize e rilasciandole su una banda di azioni.
Client dell'azione	Molto spesso rappresenta una voce di menu o un pulsante che riceve una notifica per avviare un'azione. Quando il client riceve un comando dell'utente (come il clic del mouse), avvia un'azione associata.

Tabella 8.1 Terminologia per la configurazione delle azioni

Termine	Definizione
Lista di azioni	Gestisce una lista di azioni che l'applicazione può eseguire in risposta a qualcosa fatto dall'utente.
Action Manager	Raggruppa e organizza insiemi logici di azioni che possono essere riutilizzati su componenti ActionBand. Consultare *TActionManager*.
Menu	Elenca i comandi che l'utente dell'applicazione può eseguire facendo clic su di essi. È possibile creare menu utilizzando la classe di menu ActionBand *TActionMainMenuBar* oppure utilizzando componenti multipiattaforma come *TMainMenu* o *TPopupMenu*.
Destinazione	Rappresenta l'elemento su cui un'azione esegue una certa cosa. Di solito la destinazione è un controllo, come un campo memo o un controllo dati. Non tutte le azioni richiedono una destinazione. Ad esempio, le azioni Help standard ignorano la destinazione e avviano semplicemente il sistema di aiuto.
Barra strumenti	Visualizza una riga visibile di icone pulsanti che, se si fa clic su di esse, provocano l'esecuzione di una certa azione da parte del programma, ad esempio la stampa del documento corrente. È possibile creare barre strumenti utilizzando il componente ActionBand toolbar *TActionToolBar* oppure utilizzando il componente multipiattaforma *TToolBar*.

Se si sviluppano applicazioni multipiattaforma, fare riferimento al paragrafo "Uso delle liste di azioni" a pagina 8-25.

Che cosa è un' azione

Poiché si sta sviluppando una propria applicazione, è possibile creare un insieme di azioni che è possibile utilizzare su diversi elementi dell'interfaccia utente. È possibile organizzarle per categorie che possono essere rilasciate su un menu come gruppo di azioni (ad esempio, Cut, Copy e Paste) oppure una alla volta (ad esempio, Tools | Customize).

Un'azione corrisponde a uno o più elementi dell'interfaccia utente, come comandi di menu o pulsanti della barra strumenti. Le azioni assolvono due funzioni*:* (1) rappresentano le proprietà comuni agli elementi dell'interfaccia utente, come lo stato attivo o inattivo di un controllo, e (2) rispondono alla stimolazione dei controlli, ad esempio, quando l'utente dell'applicazione fa clic su un pulsante o sceglie una voce di menu. È possibile creare un repertorio di azioni da rendere disponibili all'applicazione attraverso menu, pulsanti, barre strumenti, menu di scelta rapida, e così via.

Le azioni sono associate ad altri componenti:

- **Client***:* Uno o più client utilizzano l'azione. Il client rappresenta molto spesso una voce di menu o un pulsante (ad esempio, *TToolButton*, *TSpeedButton*, *TMenuItem*, *TButton*, *TCheckBox, TRadioButton*, e così via). Le azioni possono risiedere anche in componenti ActionBand come *TActionMainMenuBar* e *TActionToolBar*. Quando il client riceve un comando dell'utente (come il clic del mouse), avvia un'azione associata. Di solito, l'evento *OnClick* di un client è associato all'evento *OnExecute* della sua azione.

- **Destinazione**: L'azione agisce in base alla destinazione. Di solito la destinazione è un controllo, come un campo memo o un controllo dati. Gli scrittori di componenti possono creare azioni specifiche per le necessità dei controlli che essi progettano e utilizzano e quindi assemblare quelle unit per creare applicazioni più modulari. Non tutte le azioni utilizzano una destinazione. Ad esempio, le azioni Help standard ignorano la destinazione e avviano semplicemente il sistema di aiuto.

  Una destinazione può anche essere un componente. Ad esempio, i controlli per dati cambiano la destinazione in un dataset associato.

Il client influisce sull'azione–l'azione risponde quando un client stimola l'azione. L'azione influisce anche sul client–le proprietà dell'azione aggiornano dinamicamente le proprietà del client. Ad esempio, se in fase di esecuzione un'azione è disabilitata (impostando la sua proprietà *Enabled* a **false**), ogni client di quell'azione è disabilitato e viene visualizzato come disabilitato.

È possibile aggiungere, cancellare e riordinare le azioni utilizzando l'Action Manager o l'Action List editor (che vengono visualizzati facendo doppio clic su un oggetto della lista di azioni, *TActionList*). Queste azioni saranno connesse successivamente a controlli client.

Configurazione delle bande di azione

Poichè le azioni non sono in grado di conservare alcuna informazione sul proprio layout (informazoni di aspetto o di posizionamento), C++Builder dispone di bande di azioni in grado di memorizzare questi dati. Le bande di azioni sono quindi un meccanismo che permette di specificare informazioni di disposizione e un insieme di controlli. È possibile rappresentare le azioni come elementi dell'interfaccia utente, come barre strumenti e menu.

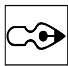
Gli insiemi di azioni vengono organizzati utilizzando l'Action Manager (*TActionManager*). È possibile utilizzare azioni standard già predisposte o creare nuove azioni personali.

Quindi si creano le bande di azioni:

- Utilizzare *TActionMainMenuBar* per creare un menu principale.
- Utilizzare *TActionToolBar* per creare una barra strumenti.

Le bande di azioni agiscono come contenitori che contengono e rappresentano insiemi di azioni. In progettazione è possibile trascinare gli elementi dall'Action Manager editor sulla banda di azioni. In esecuzione, gli utenti dell'applicazione possono anche personalizzare i menu o le barre strumenti dell'applicazione utilizzando una finestra di dialogo simile all'Action Manager editor.

Creazione di barre di strumenti e menu



Questa sezione descrive il metodo consigliato per la creazione di menu e di barre strumenti in applicazioni Windows. Nello sviluppo multipiattaforma, è necessario utilizzare *TToolBar* e componenti menu, come *TMainMenu*, e organizzarli mediante le

liste di azioni (*TActionList*). Vedere "Configurazione delle liste di azioni" a pagina 8-25.

Si utilizza l' Action Manager per generare automaticamente barre strumenti e menu principali in base alle azioni incluse nell'applicazione. L'Action Manager gestisce le azioni standard e tutte le azioni custom che sono state scritte. Quindi si creano gli elementi del'interfaccia utente in base a queste azioni e si utilizzano le bande di azioni per rappresentare i vari elementi delle azioni elementi come voci di menu o come pulsanti di una barra strumenti.

La procedura generale per la creazione di menu, barre strumenti e altre bande di azioni implica le seguenti fasi:

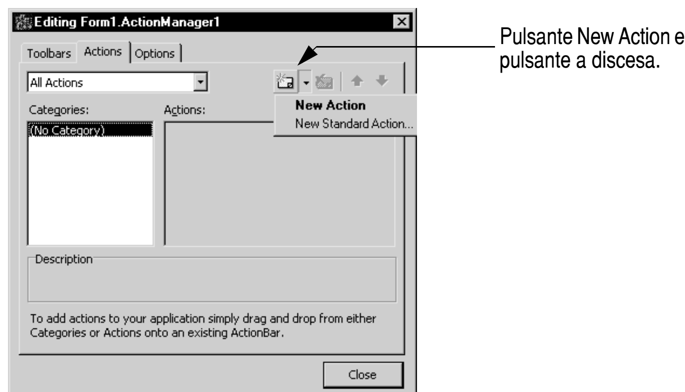
- Collocare su una scheda un componente Action Manager.
- Aggiungere azioni all'Action Manager, che le organizza per liste di azioni appropriate.
- Creare le bande di azioni (cioè, il menu o la barra strumenti) per l'interfaccia utente.
- Trascinare e rilasciare le azioni nell'interfaccia dell'applicazione.

La seguente procedura spiega in maggior dettaglio le varie fasi.

Per creare i menu e le barre strumenti utilizzando le bande di azioni:

1 Dalla pagina Additional della Component palette, trascinare e rilasciare sulla scheda un componente *Action* Manager (*TActionManager*) nel punto in cui si desidera creare la barra strumenti o il menu.
2 Se sul menu o sulla barra strumenti si vogliono delle immagini, trascinare e rilasciare sulla scheda un componente ImageList dalla pagina Win32 della Component palette. (È necessario aggiungere all'ImageList le immagini che si vogliono utilizzare oppure quelle fornito.)
3 Dalla pagina Additional della Component palette, trascinare e rilasciare sulla scheda una o più delle seguenti bande di azioni:
   - *TActionMainMenuBar* (per progettare menu principali)
   - *TActionToolBar* (per progettare barre strumenti)
4 Connettere il componente ImageList al gestore di azioni*:* con il fuoco sull'Action Manager e nell'Object Inspector, selezionare il nome dell'ImageList dalla proprietà Images.
5 Aggiungere le azioni al pannello delle azioni dell'Action Manager editor:
   - Fare doppio clic sull'Action Manager per visualizzare l'Action Manager editor.
   - Fare clic sulla freccia verso il basso accanto al pulsante New Action (il pulsante più a sinistra posto nell'angolo in alto a destra della pagina Actions, come mostrato nella Figure 8.2) e selezionare "New Action" oppure "New Standard Action". Viene mostrata una vista ad albero. Aggiungere al pannello delle azioni dell'Action Manager una o più azioni o categorie di azioni. L'Action Manager aggiungerà le azioni alla proprie liste di azioni.

Figura 8.2 Action Manager editor



6 Trascinare singole azioni o categorie di azioni dall'*Action* Manager editor e rilasciarle sul menu o sulla barra strumenti che si sta progettando.

Per aggiungere azioni definite dall'utente, creare un nuovo *TAction* utilizzando il pulsante New Action e scrivendo un gestore di evento che definisca come risponderà al momento dell'attivazione. Per i dettagli, consultare "Cosa accade quando si scatena un'azione" a pagina 8-27. Una volta che si sono definite le azioni, è possibile trascinarle sui menu o sulle barre strumenti esattamente come le azioni standard.

Aggiunta di colori, motivi o immagini a menu, pulsanti e barre di strumenti

È possibile utilizzare le proprietà *Background* e *BackgroundLayout* per specificare un colore, un motivo o una bitmap da utilizzare in una voce di menu o in un pulsante. Queste proprietà permettono anche di configurare un banner da disporre sul lato sinistro o destro di un menu.

Sfondo e layout vengono assegnati ai sottoelementi a partire dai rispettivi oggetti client di azioni. Se si desidera impostare lo sfondo degli elementi di un menu, fare clic nel Form designer sulla voce di menu che contiene gli elementi. Ad esempio, se si seleziona *File* è possibile modificare lo sfondo degli elementi visualizzati nel menu File. È possibile assegnare un colore, un motivo o una bitmap mediante la proprietà *Background* nell'Object Inspector.

Utilizzare la proprietà *BackgroundLayout* per descrivere come collocare lo sfondo sull'elemento. I colori o le immagini possono essere collocati normalmente dietro l'intestazione, espansi in modo da adattarli all'area dell'elemento o affiancati in piccoli riquadri in modo da coprire l'area.

Gli elementi con sfondi impostati a normale (blNormal), espanso (blStretch), o affiancato (blTile) vengono rappresentati con uno sfondo trasparente. Se si crea un banner, l'immagine completa viene collocata alla sinistra (blLeftBanner) o alla destra (blRightBanner) dell'elemento. È necessario accertarsi che le dimensioni siano corrette in quanto non viene espando o ristretto in modo da adattarlo all'area.

Per modificare lo sfondo di una banda di azioni (cioè, di un menu principale o di una barra strumenti), selezionare la banda di azioni e scegliere il *TActionClientBar* utilizzando l'Action Band collection editor. È possibile impostare proprietà

Background e *BackgroundLayout* per specificare un colore, un motivo o una bitmap da utilizzare nell'intera barra strumenti o nel menu.

Aggiunta di icone a menu e barre di strumenti

È possibile aggiungere icone accanto alle voci di menu o sostituire le intestazioni delle barre strumenti con icone. Le bitmap o le icone vengono organizzate utilizzando un componete *ImageList*.

1 Trascinare e rilasciare sulla scheda un componente *ImageList* prelevandolo dalla pagina Win32 della Component palette.

2 Aggiungere all'elenco delle immagini quelle che si desidera utilizzare. Fare doppio clic sull'icona *ImageList*. Fare clic su Add, individuare e selezionare le immagini che si desidera utilizzare e, al termine, fare clic su OK. In Program Files\Common Files\Borland Shared\Images sono incluse alcune immagini di esempio. (Le immagini dei pulsanti includono due versioni da utilizzare per i pulsanti attivi e non attivi.)

3 Dalla pagina Additional della Component palette, trascinare e rilasciare sulla scheda una o più delle seguenti bande di azioni:

- *TActionMainMenuBar* (per progettare menu principali)
- *TActionToolBar* (per progettare barre strumenti)

4 Connettere la lista delle immagini al gestore di azioni. Per prima cosa, spostare il fuoco sull'Action Manager. In seguito, selezionare nell'Object Inspector il nome della lista di immagini dalla proprietà *Images*, come ad esempio `ImageList1`.

5 Utilizzare Action Manager editor per aggiungere azioni all'Action Manager. È possibile associare un'immagine a un'azione impostandone la proprietà *ImageIndex* al suo numero nella lista di immagini.

6 Trascinare le singole azioni o le categorie di azioni dall'Action Manager editor e rilasciarle sul menu o sulla barra strumenti.

7 Per le barre strumenti in cui si vogliono visualizzare solo le icone, senza alcuna intestazione: selezionare la banda di azioni Toolbar e fare doppio clic sulla proprietà Items. Nell'editor della collezione, è possibile selezionare uno o più elementi e impostare le rispettive proprietà *Caption*.

8 Le immagini vengono visualizzate automaticamente sul menu o sulla barra strumenti.

Creazione di barre di strumenti e menu personalizzabili dagli utenti

Con l'Action Manager è possibile utilizzare bande di azioni per creare barre strumenti e menu personalizzabili. In esecuzione, gli utenti dell'applicazione potranno personalizzare le barre strumenti e i menu (bande di azioni) direttamente nell'interfaccia utente dell'applicazione utilizzando una finestra di dialogo di personalizzazione simile all'Action Manager editor.

Per consentire all'utente dell'applicazione di personalizzare una banda di azioni nell'applicazione:

1 Trascinare su una scheda un componente Action Manager.

2 Rilasciare i componenti banda di azioni (*TActionMainMenuBar, TActionToolBar*).

3 Fare doppio clic sull'Action Manager per visualizzare l'Action Manager editor:

- Aggiungere le azioni che si desidera utilizzare nell'applicazione. Aggiunge anche l'azione Customize, visualizzata alla fine dell'elenco delle azioni standard.
- Trascinare un componente *TCustomizeDlg* dalla pagina Additional e rilasciarlo sulla scheda e connetterlo all'*Action* Manager utilizzando la proprietà ActionManager. Si deve specificare un nome file in cui registrare le personalizzazioni fatte dall'utente.
- Trascinare le azioni sui componenti banda di azioni. (Assicurarsi di aggiungere l'azione Customize alla barra strumenti o al menu.)

4 Completare l'applicazione.

Quando si compila e si esegue l'applicazione, gli utenti potranno accedere a un comando Customize che visualizza una finestra di dialogo di personalizzazione simile all'Action Manager editor. Potranno trascinare voci di menu e creare barre strumenti utilizzando le stesse azioni fornite nell'Action Manager.

Occultamento di elementi e categorie inutilizzate nelle bande di azioni

Uno dei vantaggi derivanti dall'utilizzo delle ActionBands consiste nel fatto che gli elementi e le categorie inutilizzati possono essere nascosti all'utente. Col passare del tempo, le bande di azioni diventano personalizzate per gli utenti dell'applicazione, visualizzando solo gli elementi che essi utilizzano e nascondendo il resto. Gli elementi nascosti possono diventare nuovamente visibili mediante la pressione di un pulsante a discesa. Inoltre, l'utente può ripristinare la visibilità di tutti gli elementi della banda di azioni reimpostando le statistiche di utilizzo dalla finestra di dialogo di personalizzazione. L'occultamento degli elementi è il comportamento predefinito delle bande di azioni, ma può essere modificato in modo da evitare di nascondere singoli elementi, tutti gli elementi di una particolare raccolta (ad esempio, il menu File) o di tutti gli elementi in una certa banda di azioni.

Il gestore di azioni tiene traccia del numero di volte che un'azione è stata richiamata dall'utente, memorizzando tale numero nel campo *UsageCount* del *TActionClientItem* associato. Il gestore di azioni registra anche il numero di volte che l'applicazione è stata stata eseguita, che si potrebbe definire come il numero di sessioni, e il numero di sessione in cui una azione è stata utilizzata per l'ultima volta. Il valore di *UsageCount* viene utilizzato per individuare il numero massimo di sessioni di mancato utilizzo dell'elemento, prima che l'elemento stesso venga nascosto, il quale viene quindi confrontato con la differenza fra il numero di sessione corrente e il numero di sessione dell'ultimo utilizzo dell'elemento. Se la differenza è maggiore del

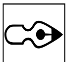
numero impostato in *PrioritySchedule*, l'elemento viene nascosto. I valori predefiniti di *PrioritySchedule* vengono visualizzati nella tabella seguente:

Tabella 8.2 Valori predefiniti della proprietà PrioritySchedule del gestore di azioni

Numero di sessioni in cui è stato utilizzato un elemento di una banda di azioni	Numero di sessioni per cui un elemento rimarrà visibile dopo il suo ultimo utilizzo
0, 1	3
2	6
3	9
4, 5	12
6-8	17
9-13	23
14-24	29
25 o più	31

È possibile disattivare l'occultamento degli elementi in progettazione. Per impedire che una certa azione (e tutte le raccolte che la contengono) venga nascosta, individuare il suo oggetto *TActionClientItem* e impostarne la proprietà *UsageCount* a –1. Per impedire l'occultamento di un'intera raccolta di elementi, come il menu *File* o anche la barra del menu principale, individuare l'oggetto *TActionClients* associato alla raccolta e impostarne la proprietà *HideUnused* a **false**.

Uso delle liste di azioni



I contenuti di questa sezione si riferiscono alla configurazione delle barre strumenti e dei menu per lo sviluppo multipiattaforma. Per sviluppo in Windows è possibile utilizzare anche i metodi descritti di seguito. Tuttavia, l'utilizzo delle bande di azioni è più semplice e offre più possibilità. Le liste di azioni saranno gestite automaticamente dall'Action Manager. Per informazioni sull'utilizzo di bande di azioni e dell'Action Manager, consultare la sezione "Organizzazione delle azioni per barre e menu" a pagina 8-17.

Le liste di azioni gestiscono una lista di azioni che l'applicazione può eseguire in risposta a qualcosa fatto dall'utente. Utilizzando oggetti action, è possibile centralizzare le funzioni eseguite dall'applicazione a partire dall'interfaccia utente. In questo modo, non solo è possibile condividere il codice comune per eseguire determinate azioni (ad esempio, quando un pulsante di una barra strumenti e voce di menu compiono la stessa operazione), ma si dispone anche di un singolo modo centralizzato per attivare e disattivare le azioni a seconda dello stato dell'applicazione.

Configurazione delle liste di azioni

La configurazione delle liste di azioni è abbastanza facile una volta comprese le operazioni di base coinvolte:

- Creazione delle lista di azioni.
- Aggiunta di azioni alla lista di azioni.
- Impostazione delle proprietà delle azioni.
- Collegamento del client all'azione.

Le operazioni di base sono spiegate in dettaglio qui di seguito:

1. Rilasciare un oggetto *TActionList* sulla scheda o sul modulo dati. (ActionList è sulla pagina Standard della Component palette.)
2. Fare doppio clic sull'oggetto *TActionList* per visualizzare l'Action List editor.
   1. Utilizzare una delle azioni predefinite elencate nell'editor: fare clic destro e scegliere New Standard Action.
   2. Le azioni predefinite sono organizzate per categorie (come Dataset, Edit, Help e Window) nella finestra di dialogo Standard Action Classes. Selezionare tutte le azioni standard che si vogliono aggiungere alla lista di azioni e fare clic su OK.

      oppure
   3. Creare una nuova azione personale: fare clic destro e scegliere New Action.
3. Impostare nell'Object Inspector le proprietà di ogni azione. (le proprietà che si impostano influiscono su ogni client dell'azione)

   La proprietà *Name* identifica l'azione e le altre proprietà ed eventi (*Caption*, *Checked*, *Enabled*, *HelpContext*, *Hint*, *ImageIndex*, *ShortCut*, *Visible* e *Execute*) corrispondono alle proprietà e agli eventi dei suoi controlli client. Le proprietà corrispondenti del client hanno di solito, ma non necessariamente, lo stesso nome della proprietà client corrispondente. Ad esempio, la proprietà *Enabled* di un'azione corrisponde alla proprietà *Enabled* di un *TToolButton*. Tuttavia, la proprietà *Checked* di un'azione corrisponde alla proprietà *Down* di un *TToolButton*.
4. Se si utilizzano le azioni predefinite, l'azione include una risposta standard che si verifica automaticamente. Se si sta creando un'azione personale, è necessario scrivere un gestore di evento che definisca come risponde l'azione quando viene scatenata. Per i dettagli, consultare "Cosa accade quando si scatena un'azione" a pagina 8-27.
5. Collegare le azioni nella lista di azioni ai client che ne hanno bisogno:
   - Fare clic sul controllo (come un pulsante o un voce di menu) sulla scheda o sul modulo dati. NellObject Inspector, la proprietà *Action* elenca le azioni disponibili.
   - Selezionare quella che si desidera.

Le azioni standard, come *TEditDelete* o *TDataSetPost*, eseguono tutte l'azione che ci si aspetta. Se occorre, è possibile consultare la Guida di riferimento in linea per i dettagli sulle modalità di funzionamento di tutte le azioni standard. Se si dovranno scrivere azioni personalizzate, sarà necessario approfondire ciò che avviene quando l'azione viene scatenata.

Cosa accade quando si scatena un'azione

Quando si scatena un evento, si verifica una serie di eventi destinati principalmente alle azioni generiche. Poi, se l'evento non gestisce l'azione, si verifica un'altra sequenza di eventi.

Risposta con eventi

Quando si fa clic su un componente client o su un controllo client, o si agisce altrimenti su di esso, si verifica una serie di eventi a cui è possibile rispondere. Ad esempio, il seguente codice illustra il gestore di evento di un'azione che commuta la visibilità di una barra strumenti quando l'azione viene eseguita:

```
void __fastcall TForm1::Action1Execute(TObject *Sender)
{
   // Toggle Toolbar1's visibility
   ToolBar1->Visible = !ToolBar1->Visible;
}
```

È possibile fornire un gestore di evento che risponda a uno dei tre diversi livelli: l'azione, la lista di azioni o l'applicazione. Ci si deve occupare di ciò solo se si utilizza una nuova azione generica invece di un'azione standard predefinita. Se si utilizzano le azioni standard non ce ne si deve occupare, perché le azioni standard hanno un comportamento incorporato che viene eseguito quando si verificano questi eventi.

I gestori di evento risponderanno agli eventi in base al seguente ordine:

- Lista di azioni
- Applicazione
- Azione

Quando l'utente fa clic su un controllo client, C++Builder chiama il metodo *Execute* dell'azione che dapprima rimanda alla lista di azioni, quindi all'oggetto Application, quindi all'azione stessa nel caso non venga gestita né dalla lista di azioni né dall'oggetto Application. Per essere più precisi, C++Builder segue questa sequenza di smistamento quando cerca un modo per rispondere all'azione di un utente:

1. Se si fornisce un gestore di evento *OnExecute* per la lista di azioni e tale gestore è in grado di gestire l'azione, l'applicazione procede.

   Il gestore di evento della lista di azioni ha un parametro di nome *Handled*, che, per impostazione predefinita, restituisce **false.** Se il gestore è stato assegnato ed è in grado di gestire l'evento restituisce **true**, e la sequenza di elaborazione termina a questo punto. Per esempio:

   ```
   void __fastcall TForm1::ActionList1ExecuteAction(TBasicAction *Action, bool &Handled)
   {
      Handled = true;
   }
   ```

   Se non si imposta *Handled* a **true** nel gestore di evento della lista di azioni, allora l'elaborazione continua.

2. Se non è stato preparato un gestore di evento *OnExecute* per la lista di azioni oppure se il gestore di evento non è in grado di gestire l'azione, viene scatenato il

gestore di evento *OnActionExecute* dell'applicazione. Se gestisce l'azione, l'applicazione procede.

L'oggetto globale *Application* riceve un evento *OnActionExecute* nel caso nessuna delle liste di azioni dell'applicazione sia in grado di gestire un evento. Come il gestore di evento *OnExecute* dell'elenco di azioni, il gestore di *OnActionExecute* ha un parametro *Handled* che, per impostazione predefinita, restituisce **false.** Se un gestore di evento è stato assegnato ed è in grado di gestire l'evento, restituisce **true** e la sequenza di elaborazioni termina a questo punto. Per esempio:

```
void __fastcall TForm1::ApplicationExecuteAction(TBasicAction *Action, bool &Handled)
{
  // Prevent execution of all actions in Application
  Handled = true;
}
```

3 Se il gestore di evento *OnExecute* dell'applicazione non gestisce l'azione, viene scatenato il gestore di evento *OnExecute* dell'azione.

È possibile utilizzare le azioni incorporate o creare proprie classi di azione che sappiano come operare su classi di destinazione specifiche (ad esempio i controlli di editing). Se non viene trovato, a nessun livello, un gestore di evento, successivamente l'applicazione prova a trovare una destinazione su cui eseguire l'azione. Se l'applicazione individua una destinazione che sa come trattarla, richiama l'azione. Per i dettagli su come l'applicazione individua una destinazione in grado di rispondere a una classe di azione predefinita, consultare il paragrafo successivo.

Come le azioni trovano le proprie destinazioni

"Cosa accade quando si scatena un'azione" a pagina 8-27 descrive il ciclo di esecuzione che si verifica quando un utente richiama un'azione. Se a nessun gestore di evento è stato assegnato il compito di rispondere all'azione, a livello di lista di azioni, di applicazione o di azione, allora l'applicazione prova a identificare un oggetto di destinazione a cui può essere affidato l'incarico.

L'applicazione cerca la destinazione utilizzando la seguente sequenza:

1. Controllo attivo*:* L'applicazione cerca come possibile destinazione dapprima se c'è un controllo attivo.
2. Scheda attiva*:* Se l'applicazione non trova un controllo attivo o se il controllo attivo non può agire come destinazione, analizza la *ActiveForm* a video.
3. Controlli sulla scheda*:* Se la scheda attiva non è una destinazione appropriata, l'applicazione analizza gli altri controlli sulla scheda attiva per trovare una destinazione.

Se non viene individuata nessuna destinazione, nel momento in cui si scatena l'evento non accade niente.

Alcuni controlli possono estendere la ricerca per cercare di rimettere la destinazione a un componente associato; ad esempio, i controlli data-aware si rivolgono al componente dataset associato. Inoltre, alcune azioni predefinite, ad esempio, quelle della finestra di dialogo File Open, non utilizzano una destinazione.

Aggiornamento delle azioni

Quando l' applicazione è inattiva, l'evento *OnUpdate* si verifica per ogni azione collegata con un controllo o con una voce di menu visualizzati in quel momento. Ciò fornisce un'opportunità alle applicazioni di eseguire il codice centralizzato per attivare e disattivare, selezionare e deselezionare, e così via. Per esempio, il seguente codice illustra il gestore di evento *OnUpdate* per un'azione che viene "selezionata" quando la barra strumenti è visibile:

```
void __fastcall TForm1::Action1Update(TObject *Sender)
{
   // Indicate whether ToolBar1 is currently visible
   ((TAction *)Sender)->Checked = ToolBar1->Visible;
}
```



Non aggiungere al gestore di evento *OnUpdate* codice con lunghi tempi di esecuzione. Questa operazione viene eseguita ogniqualvolta l'applicazione è inattiva. Se il gestore di evento impiega troppo tempo, influenzerà negativamente le prestazioni dell'intera applicazione.

Classi di azione predefinite

È possibile aggiungere all'applicazione azioni predefinite facendo clic destro sull'Action Manager e selezionando il comando New Standard Action. Verrà visualizzata la finestra di dialogo New Standard Action Classes che elenca le classi di azione predefinite e le azioni standard associate. Queste sono azioni già incluse in C++Builder e sono oggetti che eseguono automaticamente determinate azioni. Le azioni predefinite sono organizzate nelle seguenti classi:

Tabella 8.3 Classi di azioni

Classe	Descrizione
Edit	Azioni standard di editing: Utilizzate con controlli di editing come destinazione. *TEditAction* è la classe di base dei discendenti, ognuno dei quali ridefinisce individualmente il metodo *ExecuteTarget* in modo da implementare le attività di copia, taglia e incolla utilizzando la clipboard.
Format	Azioni standard di formattazione: Utilizzate con controlli rich text per impostare opzioni di formattazione del testo come grassetto, corsivo, sottolineato, barrato, e così via.*TRichEditAction* è la classe di base dei discendenti, ognuno dei quali ridefinisce individualmente i metodi *ExecuteTarget* e *UpdateTarget* in modo da implementare le attività di formattazione della destinazione.
Help	Azioni standard di Help: Utilizzate con qualsiasi destinazione. *THelpAction* è la classe di base dei discendenti, ognuno dei quali ridefinisce individualmente il metodo *ExecuteTarget* in modo da passare il comando a un sistema di Help.
Window	Azioni standard di finestra: Utilizzate con schede come destinazioni in un'applicazione MDI. *TWindowAction* è la classe di base dei discendenti, ognuno dei quali ridefinisce individualmente il metodo *ExecuteTarget* in modo da implementare le attività di disposizione, sovrapposizione, chiusura o affiancamento di finestre figlio in applicazioni MDI.

Tabella 8.3 Classi di azioni

Classe	Descrizione
File	Azioni su file*:* Utilizzate per operazioni su file come File Open, File Run o File Exit.
Search	Azioni di ricerca*:* Utilizzato con opzioni di ricerca. *TSearchAction* implementa il comportamento comune per quelle azioni che visualizzano una finestra di dialogo non modale in cui l'utente può immettere una stringa di ricerca per effettuare ricerche in un controllo di editing.
Tab	Azioni per controlli multipagina*:* Utilizzate per spostarsi fra le varie pagine di un controllo multipagina, come per i pulsanti Prev e Next di un wizard.
List	Azioni per controlli elenco*:* Utilizzate per la gestione degli elementi in un controllo list view.
Dialog	Azioni per finestre di dialogo*:* Utilizzato con componenti finestra di dialogo. *TDialogAction* implementa il comportamento comune per quelle azioni che, quando eseguite, visualizzano una finestra di dialogo. Ogni classe discendente rappresenta una finestra di dialogo specifica.
Internet	Azioni Internet*:* Utilizzato per funzioni quali la navigazione in Internet, lo scaricamento di file e l'invio di posta.
DataSet	Azioni DataSet*:* Utilizzato con componenti dataset come destinazione. *TDataSetAction* è la classe di base dei discendenti, ognuno dei quali ridefinisce individualmente i metodi *ExecuteTarget* e *UpdateTarget* in modo da implementare le attività di navigazione ed editing della destinazione. *TDataSetAction* introduce una proprietà *DataSource* che garantisce che le azioni vengano eseguite su quel dataset. Se *DataSource* è NULL, viene usato il controllo associato ai dati che in quel momento ha il fuoco.
Tools	Strumenti*:* Strumenti aggiuntivi come *TCustomizeActionBars* per la visualizzazione automatica della finestra di dialogo di personalizzazione delle bande di azioni.

Tutti gli oggetti azione sono descritti nella Guida di riferimento in linea. Per i dettagli sul loro funzionamento, consultare la Guida in linea.

Scrittura di componenti azione

È possibile anche creare proprie classi di azione predefinite. Quando si scrivono proprie classi di azione, è possibile incorporare la capacità di essere eseguite su alcune classi di oggetti di destinazione. A questo punto, è sarà possibile utilizzare le azioni custom in modo analogo a quello utilizzato per le classi di azione predefinite. Vale a dire, se l'azione è in grado di individuare e di applicare se stessa a una classe di destinazione, è possibile assegnare semplicemente l'azione a un controllo client, ed essa agirà sulla destinazione che sia necessario scrivere un gestore di evento.

Gli scrittori di componenti possono utilizzare le classi nelle unit QStdActns e DBActns come esempi per derivare le proprie classi di azione per implementare comportamenti specifici per alcuni controlli o componenti. Le classi di base di queste azioni specializzate (*TEditAction*, *TWindowAction*, e così via) di solito ridefiniscono

HandlesTarget, *UpdateTarget* e altri metodi in modo da limitare la destinazione dell'azione a una determinata classe di oggetti. Normalmente, le classi discendenti ridefiniscono *ExecuteTarget* per eseguire un compito specialistico. Questi metodi sono descritti nella tabella seguente :

Metodo	Descrizione
HandlesTarget	Chiamato automaticamente quando l'utente richiama un oggetto (come un pulsante di una barra pulsanti o una voce di menu) collegato con l'azione. Il metodo *HandlesTarget* permette all'oggetto action di indicare se è opportuna o meno l'esecuzione, utilizzando come "destinazione" l'oggetto specificato dal parametro *Target*. Per i dettagli, consultare "Come le azioni trovano le proprie destinazioni" a pagina 8-28.
UpdateTarget	Chiamato automaticamente quando l'applicazione è inattiva in modo che le azioni si possano aggiornare in funzione delle condizioni. Utilizzare al posto di *OnUpdateAction*. Per i dettagli, consultare "Aggiornamento delle azioni" a pagina 8-29.
ExecuteTarget	Chiamato automaticamente al posto di *OnExecute* quando l'azione viene scatenata in risposta all'azione di un utente (ad esempio, quando l'utente seleziona una voce di menu o preme un pulsante strumento collegato con questa azione). Per i dettagli, consultare "Cosa accade quando si scatena un'azione" a pagina 8-27.

Registrazione delle azioni

Quando si scrivono le proprie azioni, è possibile registrarle in modo da consentirne la visualizzazione nell'Action List editor. Le azioni vengono registrate e de-registrate utilizzando le routine globali nella unit Actnlist:

```
extern PACKAGE void __fastcall RegisterActions(const AnsiString CategoryName, TMetaClass* const * AClasses, const int AClasses_Size, TMetaClass* Resource);

extern PACKAGE void __fastcall UnRegisterActions(TMetaClass* const * AClasses, const int AClasses_Size);
```

Quando si chiama *RegisterActions*, le azioni che si registrano vengono visualizzate nell'Action List editor in modo che siano utilizzabili da parte delle applicazioni. È possibile indicare un nome di categoria per organizzare le azioni, oltre a un parametro *Resource* che permette di indicare i valori predefiniti delle proprietà.

Per esempio, il seguente codice registra con l'IDE le azioni standard nella unit *MyAction*:

```
namespace MyAction
{
  void __fastcall PACKAGE Register()
  {
     // code goes here to register any components and editors
      TMetaClass classes[2] = {__classid(TMyAction1), __classid(TMyAction2)};
      RegisterActions("MySpecialActions", classes, 1, NULL);
  }
}
```

Quando si chiama *UnRegisterActions*, le azioni non vengono più visualizzate nell'Action List editor.

Creazione e gestione dei menu

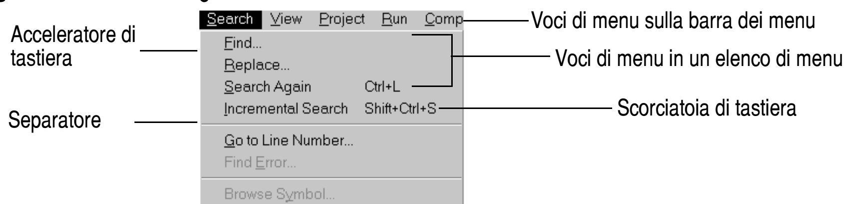
I menu forniscono agli utenti un modo facile per eseguire comandi raggruppati logicamente. Il Menu Designer di Delphi consente di aggiungere facilmente un menu, già progettato o confezionato su misura, alla scheda. È sufficiente aggiungere un componente menu alla scheda, aprire il Menu Designer e scrivere le voci menu direttamente nella finestra del Menu Designer. È possibile aggiungere o cancellare le voci di menu, o trascinarle e rilasciarle per risistemarle in fase di progettazione.

Non occorre neanche eseguire il programma per vedere i risultati–la progettazione è immediatamente visibile nella scheda, che appare proprio come apparirà in fase di esecuzione. Il codice può modificare i menu anche in fase di esecuzione, per fornire ulteriori informazioni o opzioni all'utente.

Questo capitolo spiega come usare il Menu Designer per progettare barre di menu e menu (locali) a comparsa. Esso tratta i seguenti modi di lavorare con i menu in fase di progettazione e in fase di esecuzione:

- Apertura del Menu Designer.
- Costruzione di menu.
- Editing delle voci di menu nell'Object Inspector.
- Uso del menu contestuale Menu Designer.
- Uso dei modelli di menu.
- Salvataggio di un menu come modello.
- Aggiunta di immagini alle voci di menu.

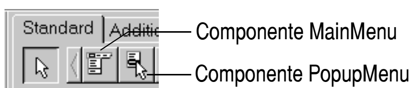
Figura 8.3 Terminologia dei menu



Apertura del Menu Designer

I menu vengono progettati utilizzando il Menu Designer. Prima di iniziare a usare il Menu Designer, è necessario aggiungere alla scheda un componente *TMainMenu* o *TPopupMenu*. Entrambi i componenti menu sono collocati sulla pagina Standard della Component palette.

Figura 8.4 Componenti MainMenu e PopupMenu



Un componente MainMenu crea un menu che viene collegato alla barra del titolo della scheda. Un componente PopupMenu crea un menu che appare quando l'utente fa clic con il tasto destro nella scheda. I menu a comparsa non hanno una barra di menu.

Per aprire il Menu Designer, selezionare un componente menu sulla scheda, e quindi scegliere:

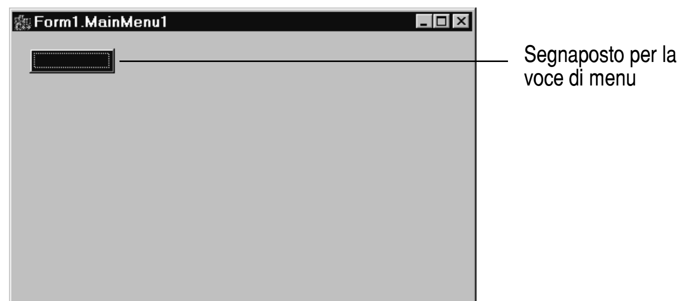
- Fare doppio clic sul componente menu.

  oppure

- Dalla pagina Properties dell'Object Inspector, selezionare la proprietà *Items*, e quindi fare doppio clic [Menu] nella colonna Value, oppure fare clic sul pulsante ellissi (...).

  Il Menu Designer viene visualizzato, con la prima voce di menu (vuota) evidenziata nel Designer, e la proprietà *Caption* selezionata nell'Object Inspector.

Figura 8.5 Menu Designer per un menu principale



Costruzione di menu

Si aggiunga un componente menu alla scheda, o alle schede, per ogni menu da includere nell'applicazione. È possibile costruire ciascuna struttura di menu completamente ex-novo, o iniziare da uno dei modelli di menu predefiniti.

Questa sezione illustra i principi per la creazioni di un menu in fase di progettazione. Per ulteriori informazioni sui modelli di menu, consultare "Uso dei modelli di menu" a pagina 8-41.

Assegnazione dei nomi dei menu

Come tutti i componenti, quando alla scheda viene aggiunto un componente menu, C++Builder gli dà un nome predefinito; per esempio, *MainMenu1*. È possibile dare al menu un nome più significativo.

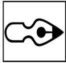
C++Builder aggiunge il nome del menu alla dichiarazione di tipo della scheda, e quindi questo nome appare nel Component list.

Assegnazione dei nomi delle voci di menu

Al contrario del componente menu stesso, occorre dare esplicitamente un nome alle voci di menu via via che vengono aggiunte alla scheda. È possibile farlo in due modi:

- Scrivere direttamente il valore per la proprietà *Name*.
- Scrivere prima il valore per la proprietà *Caption*, e lasciare a C++Builder il compito di derivare la proprietà *Name* dal caption.

  Per esempio, se viene data a una voce di menu una valore della proprietà *Caption* di *File*, C++Builder assegna alla voce di menu una proprietà *Name* di *File1*. Se viene compilata la proprietà *Name* prima di compilare la proprietà *Caption*, C++Builder lascia la proprietà *Caption* vuota finché non viene scritto un valore.



Se vengono immessi nella proprietà *Caption* caratteri non validi per gli identificatori C++, C++Builder modifica conseguentemente la proprietà *Name*. Per esempio, se si desidera che il caption inizi con un numero, C++Builder fa precedere il numero con un carattere per derivare la proprietà *Name*.

La seguente tabella mostra alcuni esempi di ciò, presupponendo che tutte le voci di menu mostrate appaiano nella stessa barra di menu.

Tabella 8.4 Esempi di caption e loro nomi derivati

Caption del componente	Nome derivato	Spiegazione
&File	File1	Rimuove la e commerciale
&File (2nd occurrence)	File2	Ordina numericamente le voci duplicate
1234	N12341	Aggiunge una lettera prima e l'ordine numerico
1234 (2nd occurrence)	N12342	Aggiungi un numero per eliminare l'ambiguità al nome derivato
$@@@#	N1	Rimuove tutti i caratteri non standard, aggiungendo una lettere prima e l'ordine numerico
- (hyphen)	N2	Ordinamento numerico della seconda occorrenza di caption senza alcun carattere standard

Come il componente menu, C++Builder aggiunge qualsiasi nome di voce di menu alla dichiarazione di tipo della scheda, e quei nomi quindi appaiono nella Component list.

Aggiunta, inserimento e cancellazione delle voci di menu

Le seguenti procedure descrivono come eseguire i principali compiti coinvolti nella costruzione della struttura del menu. Ciascuna procedura assume che sia aperta la finestra Menu Designer.

Per aggiungere voci di menu in fase di progettazione:

1. Selezionare la posizione in cui creare la voce di menu.

   Se il Menu Designer è stato appena aperto, la prima posizione sulla barra di menu è già selezionata.

2 Iniziare a scrivere per immettere il caption. O immettere prima la proprietà *Name* collocando specificamente il cursore nell'Object Inspector e immettendo un valore. In questo caso, occorre allora selezionare di nuovo la proprietà *Caption* ed immettere un valore.

3 Premere *Invio*.

   Viene selezionato il successivo placeholder per una voce di menu.

   Se prima viene immessa la proprietà *Caption*, usare i tasti cursore per ritornare alla voce di menu appena immessa. C++Builder visualizzerà di aver compilato la proprietà *Name* in base al valore immesso per il caption. (Consultare il paragrafo "Assegnazione dei nomi delle voci di menu" a pagina 8-34.)

4 Continuare l'immissione dei valori per le proprietà *Name* e *Caption* per ciascuna nuova voce da creare, o premere *Esc* per ritornare alla barra dei menu.

   Usare i tasti cursore per spostarsi dalla barra dei menu nel menu, e quindi spostarsi tra le voci dell'elenco; premere *Invio* per completare un'azione. Per ritornare alla barra dei menu, premere *Esc*.

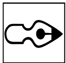
Per inserire una nuova voce di menu vuota:

1 Collocare il cursore su una voce di menu.
2 Premere *Ins*.

   Le voci di menu vengono inserite a sinistra della voce selezionata sulla barra dei menu, e sopra la voce selezionata nell'elenco del menu.

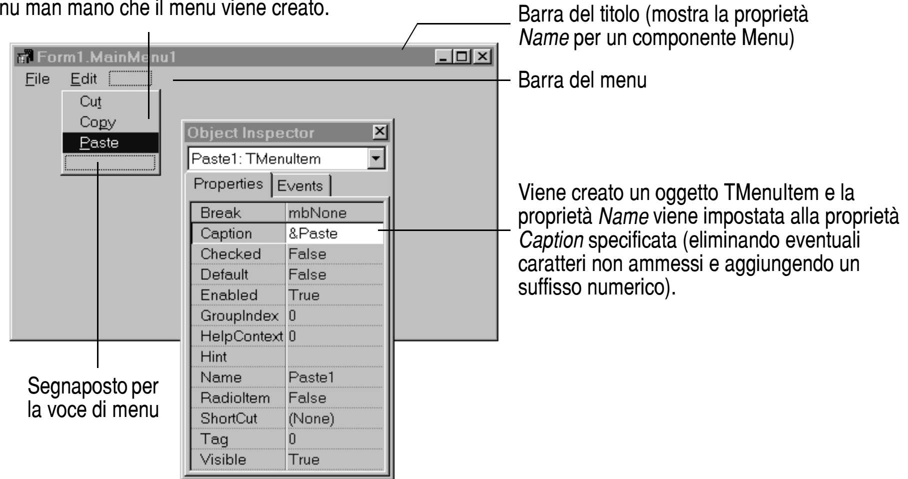
Per cancellare una voce di menu o comando:

1 Collocare il cursore sulla voce di menu da cancellare.
2 Premere *Del*.



Non è possibile cancellare il placeholder predefinito che appare sotto l'ultima voce immessa nell'elenco del menu, o vicino all'ultima voce sulla barra dei menu. Questo placeholder non appare nel menu in fase di esecuzione.

Figura 8.6 Aggiunta di voci di menu a un menu principale



Aggiunta di barre separatrici

Le barre separatrici inseriscono una riga tra le voci di menu. È possibile usare queste barre per indicare raggruppamenti all'interno dell'elenco del menu, o semplicemente per fornire un'interruzione visuale di un elenco.

Per rendere la voce di menu una barra separatrice, scrivere un trattino (–) per il caption.

Specifica dei tasti di scelta rapida e delle scorciatoie di tastiera

I tasti di scelta rapida consentono all'utente di accedere ad comando di menu dalla tastiera premendo *Alt*+ la lettera appropriata, indicata nel codice dalla e commerciale precedente. Nel menu, la lettera dopo la e commerciale appare sottolineata.

C++Builder controlla automaticamente l'esistenza di tasti di scelta rapida duplicati e li corregge in fase di esecuzione. Questo comportamento garantisce che tutti i menu generati dinamicamente in fase di esecuzione non contengano tasti di scelta rapida duplicati e che tutti gli elementi di menu abbiano un tasto di scelta rapida. È possibile disattivare questo controllo automatico impostando la proprietà *AutoHotkeys* di un elemento di menu a *maManual*.

Per specificare un tasti di scelta rapida:

- Aggiungere una e commerciale davanti alla lettera appropriata.

  Per esempio, per aggiungere un comando di menu Save con la *S* come tasto acceleratore, scrivere `&Save`.

Le scorciatoie di tastiera consentono all'utente di eseguire l'azione senza accedere direttamente al menu, scrivendo la combinazione di tasti della scorciatoia.

Per specificare una scorciatoia di tastiera:

- Usare l'Object Inspector per immettere un valore per la proprietà *ShortCut*, o selezionare una combinazione di tasti dall'elenco a discesa.

  Questo elenco è solo un sottoinsieme delle combinazioni valide che possono essere scritte.

Quando viene aggiunta una scorciatoia, essa appare vicino al titolo della voce di menu.



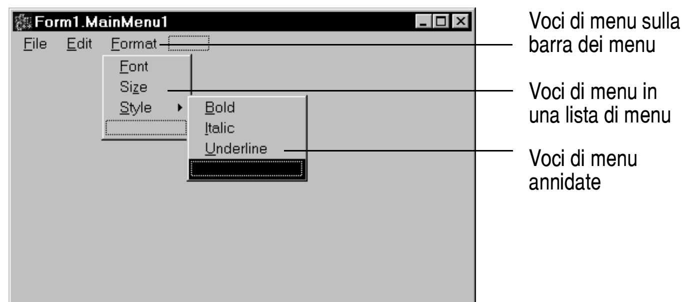
Le scorciatoie di tastiera, a differenza degli acceleratori, non sono soggette al controllo automatico per l'esistenza di duplicati. Il controllo di univocità deve essere eseguito manualmente.

Creazione di sottomenu

Molti menu di applicazioni contengono elenchi a discesa che appaiono vicino a una voce di menu per fornire ulteriori comandi collegati. Tali elenchi vengono indicati da una freccia a destra della voce di menu. C++Builder supporta tutti i livelli di sottomenu desiderati.

Organizzando la struttura dei menu in questo modo, si può salvare lo spazio verticale dello schermo. Tuttavia, per progettazioni ottimali forse è meglio non usare più di due o tre livelli nella progettazione dell'interfaccia. (Per i menu popup, potrebbe essere opportuno usare un solo sottomenu.)

Figura 8.7 Strutture di menu annidate



Per creare un sottomenu:

1. Selezionare la voce di menu sotto cui si desidera creare un sottomenu.
2. Premere *Ctrl* → per creare il primo segnaposto, o fare clic con il tasto destro e scegliere Create Submenu.
3. Scrivere un nome per la voce del sottomenu, o trascinare una voce di menu esistente su questo segnaposto.
4. Premere *Invio*, o ↓, per creare il successivo segnaposto.
5. Ripetere i punti 3 e 4 per ciascuna voce da creare nel sottomenu.
6. Premere *Esc* per ritornare al precedente livello di menu.

Creazione di sottomenu per retrocessione dei menu esistenti

È possibile creare un sottomenu inserendo una voce dalla barra dei menu (o da un modello di menu) tra le voci di menu di un elenco. Quando un menu viene spostato in una struttura di menu esistente, tutte le voci associate lo seguono, creando un menu completamente intatto. Questo vale anche per i sottomenu. Spostando una voce di menu in un sottomenu esistente si crea solo un altro livello di annidamento.

Spostamento delle voci di menu

In fase di progettazione, per spostare le voci di menu basta semplicemente trascinare e rilasciare. È possibile spostare le voci della barra dei menu, o in un altro posto dell'elenco del menu, o in un menu completamente diverso.

L'unica eccezione a ciò è di tipo gerarchico: non è possibile degradare una voce dalla barra dei menu nel suo stesso menu; né è possibile spostare una voce di menu nel suo stesso sottomenu. Però, si può spostare qualsiasi voce in un *altro* menu, indipendentemente dalla sua posizione originaria.

Durante il trascinamento, il cursore cambia forma per indicare se è possibile rilasciare la voce di menu nella nuova locazione. Quando viene spostata una voce di menu, si sposta anche qualsiasi voce al di sotto di essa.

Per spostare una voce di menu lungo la barra dei men:,

1. Trascinare la voce lungo la barra dei menu finché la punta della freccia del cursore di trascinamento punta alla nuova locazione.
2. Rilasciare il pulsante del mouse per abbandonare la voce di menu nella nuova locazione.

Per spostare una voce in un elenco di menu:

1. Trascinare la voce lungo la barra dei menu finché la punta della freccia del cursore di trascinamento punta al nuovo menu.

   Questo obbliga il menu ad aprirsi, consentendo di trascinare la voce nella sua nuova locazione.
2. Trascinare la voce di menu nell'elenco, rilasciando il pulsante del mouse per abbandonare la voce di menu nella nuova locazione.

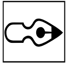
Aggiunta di immagini alle voci di menu

Le immagini possono aiutare gli utenti a navigare nei menu facendo corrispondere glyph e immagini all'azione della voce di menu, in modo analogo alle immagini della barra strumenti. È possibile aggiungere singole bitmap alle voci di menu oppure è possibile organizzare le immagini per l'applicazione in una lista di immagini e aggiungerle a un menu prelevandole dalla lista di immagini. Se nell'applicazione si utilizzano numerose bitmap delle stesse dimensioni, è utile collocarle in una lista di immagini.

Per aggiungere una singola immagine a un menu o a una voce di menu, impostarne la proprietà *Bitmap* in modo che faccia riferimento al nome della bitmap che sarà utilizzata sul menu o sulla voce di menu.

Per aggiungere un'immagine a una voce di menu utilizzando una lista di immagini:

1 Rilasciare un oggetto *TMainMenu* o *TPopupMenu* su una scheda.
2 Rilasciare un oggetto *TImageList* sulla scheda.
3 Aprire l'ImageList editor facendo doppio clic sull'oggetto *TImageList*.
4 Fare clic su Add per selezionare la bitmap o il gruppo di bitmap da usare nel menu. Fare clic su OK.
5 Impostare la proprietà *Images* dell'oggetto *TMainMenu* o *TPopupMenu* all'*ImageList* appena creato.
6 Creare le voci di menu e di sottomenu come precedentemente descritto.
7 Selezionare la voce di menu per la quale si desidera un'immagine nell'Object Inspector e impostare la proprietà *ImageIndex* al corrispondente numero dell'immagine nell'*ImageList* (il valore predefinito per *ImageIndex* è –1, che non visualizza alcuna immagine).



Usare immagini 16 per 16 pixel per una visualizzazione corretta nel menu. Anche se è possibile usare altre dimensioni, quando si usano immagini più grandi o più piccole potrebbero nascere problemi di allineamento e di coerenza.

Visualizzazione del menu

In fase di progettazione, è possibile visualizzare il menu nella scheda senza prima eseguire il codice del programma. (I menu del componente a comparsa sono visibili nella scheda in fase di progettazione, ma ciò non succede per gli stessi menu a comparsa. Usare il Menu Designer per visualizzare un menu a comparsa in fase di progettazione.)

Per visualizzare il menu:

1 Se la scheda è visibile, fare clic sulla scheda, o dal menu View, scegliere la scheda di cui si vuole visualizzare il menu.
2 Se la scheda ha più di un menu, selezionare il menu desiderato dall'elenco a discesa della proprietà *Menu* della scheda.

   Il menu appare nella scheda esattamente come apparirà quando il programma viene eseguito.

Editing delle voci di menu nell'Object Inspector

Questa sezione ha trattato come impostare alcune proprietà per le voci di menu–per esempio, le proprietà *Name* e *Caption*–usando il Menu Designer.

La sezione ha anche descritto come impostare le proprietà di una voce di menu, come *ShortCut*, direttamente nell'Object Inspector, proprio come si farebbe per qualsiasi componente selezionato nella scheda.

Quando si modifica una voce di menu usando il Menu Designer, le sue proprietà vengono ancora visualizzate nell'Object Inspector. Si può passare il fuoco all'Object Inspector e continuare lì l'editing delle proprietà della voce di menu. Oppure si può

selezionare la voce di menu dal Component list nell'Object Inspector e modificarne le proprietà senza aprire sempre il Menu Designer.

Per chiudere il Menu Designer e continuare la modifica delle voci di menu:

1 Passare il fuoco dalla finestra Menu Designer all'Object Inspector facendo clic sulla pagina Properties dell'Object Inspector.
2 Chiudere il Menu Designer coma al solito.

   Il fuoco rimane nell'Object Inspector, dove è possibile continuare l'editing delle proprietà per la voce di menu selezionata. Per modificare un'altra voce di menu, selezionarla dal Component list.

Uso del menu contestuale Menu Designer

Il menu contestuale di Menu Designer fornisce un rapido accesso ai comandi più usati del Menu Designer, e alle opzioni relative ai modelli di menu. Per ulteriori informazioni sui modelli di menu, consultare "Uso dei modelli di menu" a pagina 8-41.)

Per visualizzare il menu contestuale, fare clic con il tasto destro sulla finestra Menu Designer, o premere *Alt+F10* mentre il cursore si trova nella finestra.

I comandi del menu contestuale

La seguente tabella riporta i comandi del menu contestuale Menu Designer.

Tabella 8.5 Comandi del menu contestuale Menu Designer

Comando	Azione
Insert	Inserisce un placeholder sopra o a sinistra del cursore.
Delete	Cancella la voce di menu selezionata (e tutti i suoi sottoelementi, se ne esistono).
Create Submenu	Crea un placeholder a un livello annidato e aggiunge una freccia alla destra della voce di menu selezionata.
Select Menu	Apre un elenco di menu nella scheda corrente. Facendo doppio clic su un nome di menu si apre la finestra designer per il menu.
Save As Template	Apre la finestra di dialogo Save Template, dove è possibile salvare un menu che potrebbe essere utile in futuro.
Insert From Template	Apre la finestra di dialogo Insert Template, dove è possibile selezionare un modello da riutilizzare.
Delete Templates	Apre la finestra di dialogo Delete Templates, dove è possibile cancellare un qualsiasi modello esistente.
Insert From Resource	Apre la finestra di dialogo Insert Menu from file Resource, dove è possibile scegliere un file .rc oppure .mnu da aprire nella scheda corrente.

Passaggio da un menu all'altro in fase di progettazione

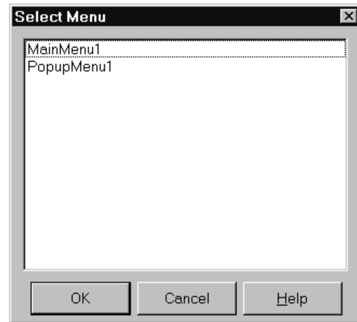
Se per la scheda occorre creare molti menu, si può usare il menu contestuale Menu Designer o l'Object Inspector per selezionare e spostarsi facilmente dall'uno all'altro.

Per usare il menu contestuale per passare da un menu all'altro di una scheda:

1 Fare clic destro nel Menu Designer e scegliere Select Menu.

  Viene visualizzata la finestra di dialogo Select Menu.

  Figura 8.8 Finestra di dialogo Select Menu



  Questa finestra di dialogo elenca tutti i menu associati alla scheda il cui menu è attualmente aperto nel Menu Designer.

2 Dall'elenco della finestra di dialogo Select Menu, scegliere il menu da visualizzare o modificare.

Per usare Object Inspector to switch between menus in a form:

1 Passare il fuoco sulla scheda da cui scegliere i menu.
2 Dal Component list, selezionare il menu da modificare.
3 Sulla pagina Properties dell'Object Inspector, selezionare la proprietà *Items* per questo menu, e quindi fare clic sul pulsante ellissi o doppio clic su [Menu].

Uso dei modelli di menu

C++Builder fornisce molti menu già progettati, o modelli di menu, che contengono i comandi usati più frequentemente.È possibile usare questi menu nella applicazioni senza modificarli (a parte la scrittura del codice), o come punto di partenza, personalizzandoli come si farebbe con un proprio menu progettato in precedenza. I modelli di menu non contengono qualsiasi codice di gestore di evento.

I modelli di menu forniti con C++Builder sono memorizzati nella sottodirectory BIN di un'installazione predefinita. Questi file hanno estensione .dmt (modelli di menu di C++Builder.

Inoltre, è possibile salvare un modello di qualsiasi menu progettato usando Menu Designer. Dopo aver salvato un menu come modello, è possibile usarlo come qualsiasi menu già progettato. Quando un particolare modello di menu non serve più, è possibile cancellarlo dall'elenco.

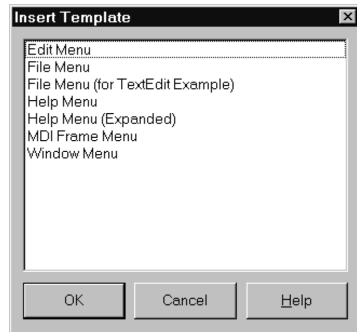
Per aggiungere un modello di menu all'applicazione:

1 Fare clic con il tasto destro sul Menu Designer e scegliere Insert From Template.

(Se non ci sono modelli, l'opzione Insert From Template nel menu contestuale appare attenuata.)

Viene aperta la finestra di dialogo Insert Template, con un elenco di modelli di menu disponibili.

Figura 8.9 Esempio di finestra di dialogo Insert Template per i menu



2 Selezionare il modello di menu da inserire, quindi premere *Invio* o scegliere OK.

Questa operazione inserisce il menu nella scheda alla posizione del cursore. Per esempio, se il cursore è su una voce di menu di un elenco, il modello di menu viene inserito sopra la voce selezionata. Se il cursore è sulla barra dei menu, il modello viene inserito a sinistra del cursore.

Per cancellare un modello di menu:

1 Fare clic con il tasto destro sul Menu Designer e scegliere Delete Templates.

(Se non ci sono modelli, l'opzione Delete Templates del menu contestuale appare attenuata.)

Viene aperta la finestra di dialogo Delete Templates, con un elenco di modelli disponibili.

2 Selezionare il modello di menu da cancellare, e premere *Del*.

C++Builder cancella il modello dall'elenco e dal disco rigido.

Salvataggio di un menu come modello

Qualsiasi menu progettato può essere salvato come modello, in modo da poterlo di nuovo. I modelli di menu possono essere usati per fornire alle applicazioni un aspetto armonioso, o come punto di partenza per ulteriori personalizzazioni.

I modelli di menu salvati vengono memorizzati nella sottodirectory BIN come file .dmt.

Per salvare un menu come modello:

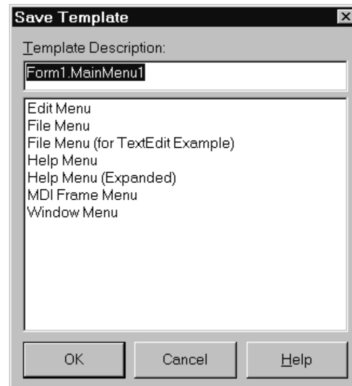
1 Progettare il menu da riutilizzare.

Questo menu può contenere tutte le voci, i comandi e i sottomenu preferiti; tutto ciò che si trova nella finestra Menu Designer attiva verrà salvato come menu riutilizzabile.

2 Fare clic con il tasto destro sul Menu Designer e scegliere Save As Template.

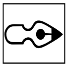
Viene visualizzata la finestra di dialogo Save Template.

Figura 8.10 Finestra di dialogo Save Template per menu



3 Nella casella di modifica Template Description, scrivere una breve descrizione per questo menu, e quindi scegliere OK.

La finestra di dialogo Save Template si chiude, salvando il progetto del menu e ritornando alla finestra Menu Designer.



La descrizione immessa viene visualizzata solo nelle finestre di dialogo Save Template, Insert Template e Delete Templates. Non è collegata alla proprietà *Name* o *Caption* per il menu.

Convenzioni per l'assegnazione dei nomi delle voci di menu e dei gestori di evento di un modello

Quando si salva un menu come modello, C++Builder non salva la sua proprietà *Name*, poiché ogni menu deve avere un nome caratteristico all'interno del campo d'azione del suo proprietario (la scheda). Tuttavia, quando in una scheda viene inserito il menu come modello usando il Menu Designer, allora C++Builder genera nuovi nomi per il menu stesso e per tutte le sue voci.

Per esempio, si supponga di salvare un menu *File* come modello. Nel menu originale, questo menu è stato chiamato *MyFile*. Se tale menu viene inserito come modello in un nuovo menu, C++Builder gli assegna il nome *File1*. Se il menu viene inserito in un menu con una voce di menu chiamata *File1*, C++Builder gli assegna il nome *File2*.

C++Builder non salva neanche tutti i gestori di evento *OnClick* associata a un menu salvato come modello, poiché non c'è alcun modo per verificare se il codice è valido nella nuova scheda. Quando viene generato un nuovo gestore di evento per la voce di menu di un modello, C++Builder genera ancora il nome del gestore di evento.

È possibile associare facilmente le voci del modello di menu ai gestori di evento *OnClick* esistenti nella scheda.

Manipolazione delle voci di menu in fase di esecuzione

Qualche volta, potrebbe essere necessario aggiungere delle voci di menu a una struttura di menu esistente mentre l'applicazione è in esecuzione, in modo da fornire ulteriori informazioni o opzioni all'utente. È possibile inserire una voce di menu usando il metodo *Add* o *Insert* della voce di menu, o in alternativa nascondere e mostrare le voci di un menu cambiando la loro proprietà *Visible*. La proprietàVisible determina se la voce viene visualizzata nel menu. Per attenuare una voce di menu senza nasconderla, usare la proprietà *Enabled*.

Per esempi che usano le proprietà *Visible* e *Enabled* della voce di menu, consultare "Disattivazione delle voci di menu" a pagina 6-10.

Nelle applicazioni MDI (Multiple Document Interface) e OLE (Object Linking and Embedding), è possibile anche fondere le voci contenute in una barra di menu esistente. La seguente sezione tratta questo argomento in dettaglio.

Fusione dei menu

Per le applicazioni MDI, come un'applicazione di esempio di editor di testi, e per le applicazioni client OLE, il menu principale dell'applicazione deve essere in grado di ricevere voci di menu da un'altra scheda o dall'oggetto server OLE. Spesso, ciò viene chiamato *fusione di menu*. È bene ricordare che la tecnologia OLE è utilizzabile solo in applicazioni Windows e non è disponibile per lo sviluppo di applicazioni multipiattaforma.

I menu da fondere vengono preparati specificando i valori di due proprietà:

- *Menu*, una proprietà della scheda
- *GroupIndex*, una proprietà delle voci del menu

Specifica del menu attivo: proprietà Menu

La proprietà *Menu* specifica il menu attivo per la scheda. Le operazioni per la fusione dei menu valgono solo per il menu attivo. Se la scheda contiene più di un componente menu, è possibile cambiare il menu attivo in fase di esecuzione impostando la proprietà *Menu* nel codice. Per esempio,

```
Form1->Menu = SecondMenu;
```

Determinazione dell'ordine delle voci di menu fuse: proprietà GroupIndex

La proprietà *GroupIndex* determina l'ordine in cui le voci di menu fuse appaiono nella barra dei menu condivisa. Le voci di menu della fusione possono sostituire quelle sulla barra del menu principale, o essere mischiate.

Il valore predefinito per *GroupIndex* è 0. Quando si specifica un valore per *GroupIndex* si devono applicare diverse regole:

- I numeri più bassi appaiono prima (più a sinistra) nel menu.

  Per esempio, impostare la proprietà *GroupIndex* a 0 (zero) per un menu che debba essere sempre il primo a sinistra, come un menu *File*. Analogamente, specificare un numero alto (non necessariamente in sequenza) per un menu che debba essere sempre il primo a destra, come un menu Help.

- Per sostituire le voci del menu principale, dare alle voci sul menu figlio lo stesso valore *GroupIndex*.

  Questa regola è applicabile ai raggruppamenti e alle singole voci. Per esempio, se la scheda principale ha una voce di menu Edit con un valore *GroupIndex* di 1, è possibile sostituirla con una o più voci dal menu della scheda figlia dando anche a questa un valore *GroupIndex* di 1.

  Fornendo alle stesse voci del menu figlio lo stesso valore *GroupIndex* si tiene il loro ordine intatto quando vengono fuse nel menu principale.

- Per inserire le voci senza sostituire quelle del menu principale, lasciare spazio nell'intervallo numerico delle voci del menu principale e "inserire" i numeri dalla scheda figlia.

  Per esempio, numerare le voci del menu principale 0 e 5, e inserire le voci del menu figlio numerandole 1, 2, 3 e 4.

Importazione dei file risorsa

C++Builder supporta menu costruiti con altre applicazioni, purché siano nel formato file risorsa standard (.RC) di Windows. Tali menu possono essere importati direttamente nel progetto C++Builder, risparmiando il tempo e la fatica di costruire di nuovo i menu che sono stati creati altrove.

Per caricare i file menu .RC esistenti:

1. Nel Menu Designer, collocare il cursore nel punto in cui far apparire il menu.

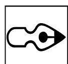
   Il menu importato può far parte di un menu in costruzione, o essere un menu completo.

2. Fare clic con il tasto destro e scegliere Insert From Resource.

   Viene visualizzata la finestra di dialogo Insert Menu From Resource.

3. Nella finestra di dialogo, selezionare il file risorsa da caricare e scegliere OK.

   Il menu appare nella finestra Menu Designer.

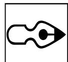


Se il file risorsa contiene più di un menu, occorre prima salvare ciascun menu come un file risorsa distinto prima di importarlo.

Progettazione di barre strumenti e di coolbar

Una *barra strumenti* è un pannello, solitamente nella parte superiore di una scheda (sotto la barra dei menu), che contiene pulsanti e altri controlli. Una *coolbar* (chiamata

anche *rebar*) è una particolare barra strumenti che visualizza i controlli su bande mobili e ridimensionabili. Se vi sono più pannelli allineati al bordo superiore della scheda, essi vengono impilati in verticale, in base alla sequenza con cui sono stati aggiunti.



Le coolbar non sono disponibili in CLX per applicazioni multipiattaforma.

Su una barra strumenti possono essere messi controlli di qualsiasi tipo. Oltre ai pulsanti, potrebbe essere opportuno mettere griglie per l'uso dei colori, barre di scorrimento, etichette, e così via.

Esistono molti modi per aggiungere una barra strumenti a una scheda:

- Collocare un pannello (*TPanel*) sulla scheda e aggiungervi i controlli (di solito pulsanti di scelta rapida).
- Usare un componente barra strumenti (*TToolBar*) invece di *TPanel*, e aggiungervi i controlli. *TToolBar* gestisce i pulsanti e altri controlli, sistemandoli in righe e regolandone automaticamente dimensioni e posizioni. Se sulla barra strumenti vengono usati controlli pulsante di strumento (*TToolButton*), *TToolBar* facilita il raggruppamento dei pulsanti per funzione e fornisce altre opzioni di visualizzazione.
- Usare un componente coolbar (*TCoolBar*) e aggiungervi i controlli. La coolbar visualizza i controlli su bande mobili e ridimensionabili indipendenti.

Il modo in cui implementare la barra strumenti dipende dall'applicazione. Il vantaggio di usare il componente Panel sta nel fatto che si ha il controllo totale dell'aspetto della barra strumenti.

Usando componenti barra strumenti e coolbar, si garantisce all'applicazione di avere l'aspetto di un'applicazione Windows, poiché vengono usati i controlli Windows nativi. Se questi controlli del sistema operativo in futuro cambiano, anche l'applicazione potrebbe cambiarli. Inoltre, dal momento che la barra strumenti e la coolbar fa affidamento sui componenti standard di Windows, l'applicazione richiede COMCTL32.DLL. Le barre strumenti e le coolbar non vengono supportate dalle applicazioni WinNT 3.51.

Nelle sezioni seguenti viene descritta la procedura per:

- Aggiungere una barra strumenti e i corrispondenti controlli pulsante di scelta rapida usando il componente barra strumenti.
- Aggiungere una barra strumenti e i corrispondenti controlli pulsante strumento usando il componente barra strumenti.
- Aggiungere una coolbar usando il componente coolbar.
- Rispondere ai clic.
- Aggiungere barre strumenti e coolbar nascoste.
- Nascondere e mostrare barre strumenti e coolbar.

Aggiunta di una barra strumenti usando un componente panel

Per aggiungere una barra strumenti a una scheda usando il componente panel:

1. Aggiungere alla scheda un componente panel (dalla pagina Standard della Component palette).
2. Impostare la proprietà *Align* del pannello a *alTop*. Quando è allineato alla parte superiore della scheda, il pannello conserva l'altezza, ma adegua la sua larghezza a quella dell'area client della scheda, anche se la finestra cambia dimensioni.
3. Aggiungere i pulsanti di scelta rapida o altri controlli al pannello.

I pulsanti di scelta rapida sono progettati per lavorare sui pannelli della barra strumenti. Un pulsante di scelta rapida, normalmente, non ha alcun caption, ma solo un piccolo grafico (chiamato glyph), che rappresenta la funzione del pulsante.

I pulsanti di scelta rapida hanno tre possibili modi di funzionamento. Essi possono

- Agire come normali pulsanti a pressione
- Attivare e disattivare quando vi si fa clic sopra
- Agire come un insieme di pulsanti radio

Per implementare i pulsanti di scelta rapida sulle barre strumenti:

- Aggiungere un pulsante di scelta rapida a un pannello della barra strumenti.
- Assegnare un glyph al pulsante di scelta rapida.
- Impostare la condizione iniziale di un pulsante di scelta rapida.
- Creare un gruppo di pulsanti di scelta rapida.
- Consentire la commutazione dei pulsanti.

Aggiunta di un pulsante di scelta rapida a un pannello

Per aggiungere un pulsante di scelta rapida a un pannello della barra strumenti, collocare il componente pulsante di scelta rapida (dalla pagina Additional della Component palette) sul pannello.

Il pannello, e non la scheda, "possiede" il pulsante di scelta rapida, in modo che spostando o nascondendo il pannello si sposta o si nasconde anche il pulsante di scelta rapida.

L'altezza predefinita del pannello è 41, mentre quella dei pulsanti di scelta rapida è 25. Se la proprietà *Top* di ciascun pulsante viene impostata a 8, essi verranno centrati verticalmente. L'impostazione predefinita della griglia blocca automaticamente il pulsante di scelta rapida in quella posizione verticale.

Assegnazione del glifo a un pulsante di scelta rapida

Ogni pulsante di scelta rapida ha bisogno di un'immagine grafica, detta anche glifo, per indicare all'utente la funzione svolta. Se viene fornita solo un'immagine, il pulsante di scelta rapida manipola quell'immagine per indicare che è stato premuto, non premuto, selezionato o disattivato. Se si preferisce, è possibile fornire anche specifiche immagini distinte per ciascuno stato.

Normalmente, i glifi vengono assegnati ai pulsanti di scelta rapida in fase di progettazione, anche se è possibile assegnare glifi differenti in fase di esecuzione.

Per assegnare un glifo a un pulsante di scelta rapida in fase di progettazione:

1 Selezionare il pulsante di scelta rapida.
2 Nell'Object Inspector, selezionare la proprietà *Glyph*.
3 Fare doppio clic sulla colonna Value a fianco di *Glyph* per aprire il Picture Editor e selezionare la bitmap desiderato.

Impostazione della condizione iniziale di un pulsante di scelta rapida

I pulsanti di scelta rapida sfruttano il proprio aspetto per dare all'utente indizi relativi al loro stato e alla loro funzione. Poiché non hanno alcuna etichetta, è importante usare le parti visuali corrette per aiutare gli utenti.

La Tabella 8.6 elenca alcune azioni che è possibile impostare per cambiare l'aspetto di un pulsante di scelta rapida:

Tabella 8.6 Impostazione dell'aspetto di pulsanti scelta rapid a

Per fare in modo che un pulsante di scelta rapida:	Nella barra strumenti, impostare:
Appaia premuto	La proprietà *GroupIndex* a un valore diverso da zero e la sua proprietà *Down* a **true**.
Appaia disattivato	La proprietà *Enabled* a **false**.
Abbia un margine sinistro	La proprietà *Indent* a una valore maggiore di 0.

Se l'applicazione ha uno strumento di disegno predefinito, essere certi che il suo pulsante sulla barra strumenti è premuto quando l'applicazione viene avviata. Per eseguire questa operazione, impostare la sua proprietà *GroupIndex* a un valore diverso da zero e la sua proprietà *Down* a **true**.

Creazione di un gruppo di pulsanti di scelta rapida

Spesso, una serie di pulsanti di scelta rapida rappresenta un insieme di scelte che si escludono a vicenda. In questo caso, occorre riunire i pulsanti in un gruppo, in modo che il clic su un qualsiasi pulsante del gruppo obblighi gli altri pulsanti a risalire.

Per associare un qualsiasi numero di pulsanti di scelta rapida in un gruppo, assegnare lo stesso numero a ciascuna proprietà *GroupIndex* del pulsante di scelta rapida.

Il modo più facile per compiere questa operazione è quello di selezionare tutti i pulsanti desiderati nel gruppo, e, con l'intero gruppo selezionato, impostare *GroupIndex* a un valore caratteristico.

Come consentire la commutazione dei pulsanti

Qualche volta, si vuole avere la possibilità di fare clic su un pulsante già premuto di un gruppo e riportarlo su, non lasciando nel gruppo alcun pulsante premuto. Un tale pulsante viene chiamato commutatore. Usare *AllowAllUp* per creare un pulsante di

gruppo che agisce come un commutatore*:* facendo clic su esso una prima volta, rimane giù; facendo clic di nuovo, ritorna su.

Perché un pulsante di scelta rapida di gruppo diventi un commutatore, impostare la sua proprietà *AllowAllUp* a **true**.

Impostando *AllowAllUp* a **true** per tutti i pulsanti di scelta rapida di un gruppo, si imposta automaticamente lo stesso valore di proprietà per tutti i pulsanti del gruppo. Ciò consente al gruppo di agire come un normale gruppo, con un solo pulsante premuto alla volta, ma consente anche ai pulsanti di essere contemporaneamente tutti su.

Aggiunta di una barra strumenti usando il relativo componente

Il componente barra strumenti (*TToolBar*) offre caratteristiche di gestione e di visualizzazione dei pulsanti che i componenti pannello non hanno. Per aggiungere una barra strumenti a una scheda usando questo componente:

1. Aggiungere alla scheda un componente barra strumenti (dalla pagina Win32 della Component palette). La barra strumenti viene allineata automaticamente alla parte superiore della scheda.
2. Aggiungere alla barra i pulsanti strumento o altri controlli.

I pulsanti strumento sono progettati per lavorare sui componenti barra strumenti. Come i pulsanti di scelta rapida, i pulsanti strumento possono:

- Agire come normali pulsanti a pressione.
- Attivare e disattivare quando vi si fa clic sopra.
- Agire come un insieme di pulsanti radio.

Per implementare i pulsanti strumento su una barra strumenti:

- Aggiungere un pulsante strumento
- Assegnare le immagini ai pulsanti strumento
- Impostare l'aspetto dei pulsanti strumento
- Creare un gruppo di pulsanti strumento
- Consentire la commutazione di pulsanti strumento

Aggiunta di un pulsante strumento

Per aggiungere un pulsante strumento a una barra strumenti, fare clic con il tasto destro sulla barra strumenti e scegliere New Button.

La barra strumenti "possiede" il pulsante strumento, così che, spostando o nascondendo la barra strumenti, si sposta o si nasconde anche il pulsante. Inoltre, tutti i pulsanti strumento sulla barra strumenti mantengono automaticamente l'altezza e la larghezza. Sulla barra strumenti è possibile rilasciare altri controlli dalla Component palette, e manterranno automaticamente un'altezza uniforme. I controlli, inoltre, vanno automaticamente a capo e iniziano una nuova riga quando la dimensione orizzontale della barra strumenti non è sufficiente a contenerli.

Assegnazione di immagini ai pulsanti strumento

Ogni pulsante strumento ha una proprietà *ImageIndex* che determina quale immagine appare su di esso in fase di esecuzione. Se al pulsante strumento viene fornita solo un'immagine, il pulsante manipola quell'immagine per indicare che il pulsante è disattivato. Per assegnare le immagini ai pulsanti strumento in fase di progettazione:

1. Selezionare la barra strumenti sulla quale devono apparire i pulsanti.
2. Nell'Object Inspector, assegnare un oggetto *TImageList* alla proprietà *Images* della barra strumenti. Un elenco di immagini è una raccolta di icone o bitmap con le stesse dimensioni.
3. Selezionare un pulsante strumento.
4. Nell'Object Inspector, assegnare un intero alla proprietà *ImageIndex* del pulsante strumento che corrisponde all'immagine dell'elenco da assegnare al pulsante.

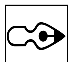
È possibile specificare anche immagini distinte che appaiono sui pulsanti strumento quando vengono disattivati e quando sono sotto il puntatore del mouse. Per eseguire questa operazione, assegnare immagini distinte alle proprietà *DisabledImages* e *HotImages* della barra strumenti.

Impostazione dell'aspetto di un pulsante strumento e delle condizioni iniziali

La Tabella 8.7 elenca alcune azioni che è possibile impostare per cambiare l'aspetto di un pulsante strumento:

Tabella 8.7 Impostazione dell'aspetto di pulsanti strumento

Per fare in modo che un pulsante strumento:	Nella barra strumenti, impostare
Appaia premuta	(La proprietà *Style* (su un pulsante strumento) a tbsChecke e la proprietà *Down* a **true**.
Appaia disattivato	La proprietà *Enabled* a **false**.
Abbia un margine sinistro	La proprietà *Indent* a una valore maggiore di 0.
Appaia con i bordi "pop-up", rendendo perciò la barra strumenti trasparente	La proprietà *Flat* a **true**.



L'uso della proprietà *Flat* di *TToolBar* richiede la versione 4.70 o successive di COMCTL32.DLL.

Per forzare una nuova riga di controlli dopo un determinato pulsante, strumento, selezionare il pulsante strumento che deve apparire per ultimo nella riga e impostare la sua proprietà *Wrap* a **true**.

Per disattivare la caratteristica auto-wrap di una barra strumenti, impostare la proprietà *Wrapable* della barra strumenti a **false**.

Creazione di gruppi di pulsanti strumento

Per creare un gruppo di pulsanti strumento, selezionare i pulsanti da associare e impostare la loro proprietà *Style* a *tbsCheck*; quindi impostare la loro proprietà *Grouped* a **true.** La selezione di un pulsante strumento di gruppo obbliga gli altri

pulsanti del gruppo a pop up, che è un utile modo di rappresentare un insieme di scelte che si escludono a vicenda.

Una qualsiasi sequenza ininterrotta di pulsanti strumento adiacenti con *Style* impostata a *tbsCheck* e *Grouped* impostata a **true** forma un singolo gruppo. Per spezzare un gruppo di pulsanti strumento, separare i pulsanti con un qualsiasi:

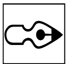
- Pulsante strumento la cui proprietà *Grouped* è **false**.
- Pulsante strumento la cui proprietà *Style* non è impostata a *tbsCheck*. Per creare spazi o divisioni sulla barra strumenti, aggiungere un pulsante strumento la cui proprietà *Style* è *tbsSeparator* o *tbsDivider*.
- Un altro controllo che non sia un pulsante strumento.

Come consentire pulsanti strumento commutati

Usare *AllowAllUp* per creare un pulsante strumento di gruppo che agisce come un commutatore: facendo clic su esso una prima volta, rimane giù; facendo clic di nuovo, ritorna su.Perché un pulsante strumento di gruppo diventi un commutatore, impostare la sua proprietà *AllowAllUp* a **true**.

Come i pulsanti di scelta rapida, impostando *AllowAllUp* a **true** per tutti i pulsanti strumento di un gruppo, si imposta automaticamente lo stesso valore di proprietà per tutti i pulsanti del gruppo.

Aggiunta di un componente coolbar



Il componente *TCoolBar* richiede la versione 4.70 o successiva di COMCTL32.DLL e non è disponibile in CLX.

Il componente coolbar (*TCoolBar*) – detto anche *rebar* – visualizza controlli con finestra su bande mobili e ridimensionabili in modo indipendente. L'utente può posizionare le bande trascinando i grip di ridimensionamento sul lato sinistro di ciascuna banda.

Per aggiungere una coolbar a una scheda in un'applicazione Window:

1. Aggiungere alla scheda un componente coolbar (dalla pagina Win32 della Component palette). La coolbar viene allineata automaticamente alla parte superiore della scheda.
2. Aggiungere alla barra i controlli windowed dalla Component palette.

Solo i componenti della VCL che discendono da *TWinControl* sono controlli con finestra. È possibile aggiungere alla coolbar controlli grafici, come etichette o pulsanti, ma non appariranno su bande distinte.

Impostazione dell'aspetto della coolbar

Il componente coolbar offre molte utili opzioni di configurazione. La Tabella 8.8 elenca alcune azioni che è possibile impostare per cambiare l'aspetto di un pulsante strumento:

Tabella 8.8 Impostazione dell'aspetto di un pulsante cool

Per fare in modo che la coolbar:	Nella barra strumenti, impostare:
Venga ridimensionata automaticamente per alloggiare le bande che contiene	La proprietà *AutoSize* a **true**.
Mantenga le bande a un'altezza uniforme	La proprietà *FixedSize* a **true**.
Venga orientata verticalmente invece che orizzontalmente	La proprietà *Vertical* a **true.** Questa operazione cambia l'effetto della proprietà *FixedSize*.
Impedisca alle proprietà *Text* delle bande la visualizzazione in fase di esecuzione	La proprietà *ShowText* a **false.** Ciascuna banda di una coolbar ha una sua proprietà *Text*.
Rimuova il bordo che la circonda	La proprietà *BandBorderStyle* a *bsNone*.
Impedisca agli utenti di cambiare l'ordine delle bande in fase di esecuzione. (L'utente può sempre spostare e ridimensionare le bande.)	*FixedOrder* a **true**.
Crei un'immagine per lo sfondo della coolbar	La proprietà *Bitmap* all'oggetto *TBitmap*.
Scelga un elenco di immagini da visualizzare a sinistra di qualsiasi banda	La proprietà *Images* all'oggetto *TImageList*.

Per assegnare immagini alle singole bande, selezionare la coolbar e fare doppio clic sulla proprietà *Bands* nell'Object Inspector. Quindi selezionare una banda e assegnare un valore alla sua proprietà *ImageIndex*.

Risposta ai clic

Quando l'utente fa clic su un controllo, come un pulsante su una barra strumenti, l'applicazione genera un evento *OnClick* a cui è possibile rispondere con un gestore di evento. Poiché *OnClick* è l'evento predefinito dei pulsanti, è possibile generare uno scheletro di gestore di evento facendo doppio clic sul pulsante in fase di progettazione.

Assegnazione di un menu a un pulsante strumento

Se viene usata una barra strumenti (*TToolBar*) con pulsanti strumento (*TToolButton*), è possibile associare un menu a uno specifico pulsante:

1. Selezionare il pulsante strumento.
2. Nell'Object Inspector, assegnare un menu a comparsa (*TPopupMenu*) alla proprietà *DropDownMenu* del pulsante strumento.

Se la proprietà *AutoPopup* del menu è impostata a **true**, apparirà automaticamente quando il pulsante viene premuto.

Aggiunta di barre strumenti nascoste

Le barre strumenti non devono essere visibili tutte le volte. Infatti, spesso è comodo avere a disposizione un gran numero di barre strumenti, ma mostrarle solo quando l'utente desidera usarle. Altrettanto spesso, quindi, viene creata una scheda che ha molte barre strumenti, ma tutte o in parte nascoste.

Per creare una barra strumenti nascosta:

1. Aggiungere alla scheda un componente barra strumenti, coolbar o pannello.
2. Impostare la proprietà *Visible* del componente a **false**.

Anche se la barra strumenti rimane visibile in fase di progettazione in modo da poterla modificare, rimane nascosta in fase di esecuzione finché l'applicazione non la rende specificamente visibile.

Come nascondere e mostrare le barre strumenti

Spesso, è opportuno che un'applicazione abbia barre strumenti multiple, ma non si vuole ingombrare la scheda visualizzandole tutte insieme. Oppure, si può lasciare agli utenti la possibilità di decidere se visualizzare le barre strumenti. Come tutti i componenti, le barre strumenti possono essere mostrate o nascoste, secondo le necessità, in fase di esecuzione.

Per nascondere o mostrare una barra strumenti in fase di esecuzione, impostare la sua proprietà *Visible,* rispettivamente, a **false** o **true.** Normalmente, questa operazione viene eseguita in risposta a particolari eventi dell'utente o a modifiche nel modo operativo dell'applicazione. Per eseguire questa operazione, normalmente c'è un pulsante close su ciascuna barra strumenti. Quando l'utente fa clic su quel pulsante, l'applicazione nasconde la corrispondente barra strumenti.

È anche possibile fornire strumenti per la commutazione della barra strumenti. nel seguente esempio, una barra strumenti di penne viene commutata da un pulsante sulla barra strumenti principale. Poiché ciascun clic preme o rilascia il pulsante, un gestore di evento *OnClick* può mostrare o nascondere al barra strumenti Pen a seconda che il pulsante sia su o giù.

```
void __fastcall TForm1::PenButtonClick(TObject *Sender)
{
  PenBar->Visible = PenButton->Down;
}
```

Capitolo

9

Tipi di controlli

I controlli sono componenti visuali che aiutano a progettare l'interfaccia utente.

Questo capitolo descrive i vari controlli che è possibile utilizzare, tra cui controlli di testo, controlli di input, pulsanti, controlli elenco, controlli di raggruppamento, controlli di visualizzazione, griglie, editor di valori di 'elenco e controlli grafici.

Per la creazione di un controllo grafico consultare il Capitolo 54, "Creazione di un controllo grafico". Per sapere come implementare tali controlli vedere il Capitolo 6, "Uso dei controlli".

Controlli di testo

Molte applicazioni utilizzano controlli di testo per mostrare il testo all'utente. È possibile utilizzare:

- Controlli di editing, che permettono all'utente di aggiungere testo.

Utilizzare questo componente:	Per consentire all'utente di:
TEdit	Modificare una singola riga di testo.
TMemo	Modificare più righe di testo.
TMaskEdit	Immettere un testo in modo conforme a un particolare formato, come un codice postale oppure un numero di telefono.
TRichEdit	Modificare più righe di testo utilizzando il formato rich text (solo per la VCL).

- Controlli per la visualizzazione del testo ed etichette, che non consentono all'utente di aggiungere testo:

Utilizzare questo componente:	Per consentire all'utente di:
TTextBrowser	Visualizzare un file di testo o una semplice pagina HTML che gli utenti possono scorrere.
TTextViewer	Visualizzare un file di testo o una semplice pagina HTML. Gli utenti possono far scorrere la pagina o fare clic sui collegamenti per visualizzare altre pagine e immagini.
TLCDNumber	Visualizzare informazioni numeriche con un formato digitale.
TLabel	Visualizzare del testo in un controllo senza finestra.
TStaticText	Visualizzare testo in un controllo con finestra.

Controlli di editing

I controlli di editing mostrano del testo all'utente e gli consentono di immettere del testo. Il tipo di controllo utilizzato a questo scopo dipende dalle dimensioni e dal formato delle informazioni.

TEdit e *TMaskEdit* sono semplici controlli di testo che includono una casella di modifica per una singola linea di testo in cui è possibile inserire informazioni. Quando la casella di modifica riceve il fuoco, viene visualizzato un punto di inserimento lampeggiante.

È possibile includere del testo nella casella di modifica assegnando un valore di stringa alla proprietà *Text*. L'aspetto del testo nella casella di modifica viene controllato assegnando dei valori alla proprietà *Font*. È possibile specificare il tipo di carattere, le dimensioni, il colore e gli attributi del carattere. Gli attributi influiscono su tutto il testo nella casella di modifica e non possono essere applicati a singoli caratteri.

Una casella di modifica può essere progettata in modo che modifichi le proprie dimensioni a seconda delle dimensioni del font utilizzato. L'operazione è possibile impostando la proprietà *AutoSize* a **true**. È possibile limitare il numero di caratteri che una casella di modifica può contenere assegnando un valore alla proprietà *MaxLength*.

TMaskEdit è uno speciale controllo per l'editing che convalida il testo immesso a fronte di una maschera che codifica i possibili formati validi che un testo può assumere. La maschera può anche formattare il testo che viene mostrato all'utente.

TMemo e *TRichEdit* permettono all'utente di aggiungere diverse righe di testo.

Proprietà dei controlli di editing

Di seguito sono elencate alcune importanti proprietà dei controlli testo:

Tabella 9.1 Proprietà dei controlli di testo

Proprietà	Descrizione
Text	Determina il testo che viene visualizzato nel controllo casella di modifica o memo.
Font	Controlla gli attributi del testo scritto nel controllo casella di modifica o memo.
AutoSize	Permette alla casella di modifica di modificare dinamicamente la propria altezza a seconda del font attualmente selezionato.
ReadOnly	Specifica se all'utente è permesso modificare il testo.
MaxLength	Limita il numero di caratteri nei controlli di solo testo.
SelText	contiene la porzione di testo selezionata (evidenziata) al momento.
SelStart, SelLength	Indicano la posizione e la lunghezza della porzione di testo.

Controlli memo e rich edit

Entrambi i controlli *TMemo* e *TRichEdit* gestiscono più linee di testo.

I controlli rich edit sono inclusi solo nella VCL.

TMemo è un altro tipo di casella di modifica che gestisce più righe di testo. Le righe in un controllo memo possono estendersi oltre il margine destro della casella di modifica o possono passare automaticamente sulla riga successiva. L'avanzamento automatico alla riga successiva viene controllato utilizzando la proprietà *WordWrap*.

Il componente TRichedit è un controllo per campi memo che supporta il formato rich text, la stampa, la ricerca e il drag-and-drop di testo. Consente di specificare le proprietà per il tipo di carattere, l'allineamento, i fermi di tabulazione, i rientri e la numerazione.

Oltre alle proprietà che tutti i controlli di editing possiedono, i controlli memo e rich edit includono altre proprietà, come le seguenti:

- *Alignment* specifica come viene allineato il testo (a sinistra, a destra oppure al centro) nel componente.
- La proprietà *Text* contiene il testo nel controllo. L'applicazione può verificare se il testo è stato modificato controllando la proprietà *Modified*.
- *Lines* contiene il testo come un elenco di stringhe.
- *OEMConvert* determina se il testo viene convertito temporaneamente da ANSI a OEM al momento dell'immissione. Questa proprietà risulta utile per convalidare nomi di file (solo VCL).
- *WordWrap* determina se il testo andrà a capo automaticamente una volta raggiunto il margine destro.
- *WantReturns* determina se l'utente può inserire nel testo caratteri di ritorno a capo forzato.
- *WantTabs* determina se l'utente può inserire nel testo caratteri di tabulazione.

- *AutoSelect* determina se il testo viene selezionato automaticamente (evidenziato) non appena il controllo diviene attivo.

In esecuzione, è possibile selezionare tutto il testo presente nel memo usando il metodo *SelectAll*.

Controlli per la visualizzazione del testo (solo per la CLX)

I controlli di visualizzazione testo mostrano il testo ma sono a sola lettura. *TTextViewer* agisce come un semplice visualizzatore in modo che gli utenti possano leggere e scorrere documenti. Con *TTextBrowser*, gli utenti possono anche fare clic sui collegamenti per navigare ad altri documenti e ad altre parti dello stesso documento. I documenti visitati sono memorizzati in una cronistoria, che può essere percorsa utilizzando i metodi *Backward*, *Forward* e *Home*. *TTextViewer* e *TTextBrowser* sono utilizzati per visualizzare testo basato su HTML o per implementare un sistema di Guida basato su HTML.

Oltre alle stesse proprietà di TtextViewer, *TTextBrowser* ha anche la proprietà *Factory*. *Factory* determina l'oggetto factory MIME utilizzato per determinare i tipi di file per le immagini incorporate. Ad esempio, è possibile associare le estensioni dei nome di file—come .txt, .html e .xml—con i tipi MIME e fare in modo che factory carichi questi dati nel controllo.

Utilizzare la proprietà *FileName* per aggiungere un file di testo, come .html, che sarà visualizzato nel controllo in esecuzione.

Etichette

Le etichette (*TLabel* e *TStaticText* (solo per la VCL) visualizzano un testo e di solito sono poste accanto ad altri controlli. Si colloca un'etichetta su una scheda quando è necessario identificare o aggiungere un'annotazione a un altro componente, ad esempio una casella di testo, o quando si desidera aggiungere del testo a una scheda. Il componente etichetta standard, *TLabel*, è un controllo non basato su finestra (non basato su widget in CLX), e pertanto non può ricevere il fuoco; nel caso occorra un'etichetta con un handle di finestra, utilizzare invece *TStaticText*.

Di seguito sono elencate alcune proprietà del componente Label:

- *Caption* contiene la stringa del testo dell'etichetta.
- *Font*, *Color*, e altre proprietà determinano l'aspetto dell'etichetta. Ogni etichetta può utilizzare un solo tipo di carattere, dimensione e colore.
- *FocusControl* collega l'etichetta a un altro controllo sulla scheda. Se *Caption* include un tasto acceleratore, quando l'utente preme tale tasto, il controllo specificato da *FocusControl* riceve fuoco.
- *ShowAccelChar* determina se l'etichetta può visualizzare un carattere acceleratore sottolineato. Se *ShowAccelChar* è **true**, qualsiasi carattere preceduto da un carattere "e commerciale" (&) appare sottolineato e attiva un tasto acceleratore.

- *Transparent* determina se gli elementi posti sotto l'etichetta (come gli elementi grafici) sono visibili o meno.

Di solito le etichette servono per visualizzare del testo statico a sola lettura che non può essere modificato dall'utente dell'applicazione. In fase di esecuzione, l'applicazione può modificare il testo assegnando un nuovo valore alla proprietà *Caption*. Per aggiungere a una scheda un oggetto testo che l'utente possa far scorrere o modificare, utilizzare *TEdit*.

Controlli specializzati per l'input

I componenti seguenti forniscono modi aggiuntivi per catturare l'input.

Utilizzare questo componente:	Per consentire all'utente di:
TScrollBar	Selezionare valori in un intervallo continuo
TTrackBar	Selezionare valori in un intervallo continuo (impatto visivo più efficace rispetto a una barra di scorrimento)
TUpDown	Selezionare un valore da un controllo selettore associato a un componente di modifica (solo VCL)
THotKey	Immettere sequenze di tastiera con *Ctrl*/*Shift*/*Alt* (solo VCL)
TSpinEdit	Selezionare un valore da un widget con casella di selezione (solo CLX)

Barre di scorrimento

Il componente scrollbar crea una barra di scorrimento che è possibile utilizzare per scorrere i contenuti di una finestra, di una scheda o di un altro controllo. Nel gestore di evento di *OnScroll* viene scritto il codice che determina come si comporta il controllo quando l'utente sposta il cursore della barra di scorrimento.

Il componente scrollbar non è utilizzato molto spesso, perché molti componenti visuali includono proprie barre di scorrimento e non richiedono scrittura di codice aggiuntivo. Per esempio, *TForm* ha le proprietà *VertScrollBar* e *HorzScrollBar* che consentono di configurare automaticamente le barre di scorrimento sulla scheda. Per creare una regione con scorrimento all'interno di una scheda utilizzare il componente *TScrollBox*.

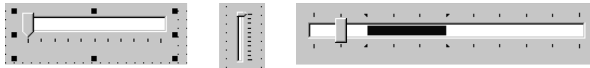
Barre di regolazione

Un componente trackbar consente di impostare un valore intero in un intervallo continuo. Risulta utile per regolare proprietà quali colore, volume e luminosità. L'utente sposta l'indicatore a scorrimento trascinandolo in una particolare posizione oppure facendo clic all'interno della barra.

- Utilizzare le proprietà *Max* e *Min* per impostare i limiti superiore e inferiore della barra di regolazione.

- Utilizzare *SelEnd* e *SelStart* per evidenziare un intervallo di selezione. Vedere la Figure 9.1.
- La proprietà *Orientation* determina se la barra di regolazione è verticale oppure orizzontale.
- Per impostazione predefinita, una barra di regolazione presenta una serie di tacche nella sua parte inferiore. Utilizzare la proprietà *TickMarks* per modificarne la posizione. Per controllare gli intervalli tra le tacche, utilizzare la proprietà *TickStyle* e il metodo *SetTick*.

Figure 9.1 Tre viste del componente track bar



- *Position* imposta una posizione predefinita per la barra di regolazione e ne traccia la posizione in esecuzione.
- Per impostazione predefinita, l'utente può spostare il cursore alla tacca precedente o successiva usando il tasti cursore Su e Giù. Impostare *LineSize* per modificare tale incremento.
- Impostare *PageSize* per determinare il numero di tacche di spostamento conseguenti alla pressione dei tasti *PgSù* e *PgGiù*.

Controlli up-down (solo VCL)

Un controllo up-down (*TUpDown*) è formato di una coppia di pulsanti freccia che consentono agli utenti di modificare un valore intero mediante incrementi fissi. Il valore corrente è dato dalla proprietà *Position*; l'incremento assunto per impostazione predefinita è pari a 1, ed è specificato dalla proprietà *Increment*. Utilizzare la proprietà *Associate* per associare al controllo di selezione un altro componente (come un controllo di editing.

Controlli spin edit (solo CLX)

Un controllo spin edit (casella di selezione) (*TSpinEdit*) viene anche definito come widget up-down, little arrows widget o pulsante a rotazione. Questo controllo permette all'utente di modificare un valore intero mediante incrementi fissi, o facendo clic sui pulsanti freccia, verso l'alto o verso il basso, per incrementare o decrementare il valore attualmente visualizzato oppure immettendo il valore direttamente nella casella di selezione.

Il valore corrente è contenuto nella proprietà *Value* ; l'incremento, che per impostazione predefinita è pari a 1, viene specificato mediante la proprietà *Increment*.

Controlli Hot key (solo VCL)

Utilizzare il componente hotkey (*THotKey*) per assegnare una scorciatoia di tastiera in grado di trasferire il fuoco a un qualsiasi controllo. La proprietà *HotKey* contiene la

combinazione di tasti corrente e la proprietà *Modifiers* determina quali tasti sono disponibile per *HotKey*.

Il componente hot key può essere assegnato come proprietà *ShortCut* a una voce di menu. Quindi, quando un utente immette la combinazione di tasti specificata dalle proprietà *HotKey* e *Modifiers* , Windows attiva la voce di menu.

Controlli splitter

Uno splitter (*TSplitter*) collocato tra controlli allineati consente all'utente di ridimensionare i controlli. Utilizzato con componenti quali pannelli e caselle di gruppo, gli splitter consentono di suddividere una scheda in diversi pannelli con più controlli su ciascun pannello.

Dopo avere collocato su una scheda un pannello o un altro controllo, aggiungere uno splitter con lo stesso allineamento del controllo. L'ultimo controllo dovrebbe essere allineato al client, in modo da occupare lo spazio rimante quando si ridimensionano gli altri componenti. Per esempio, è possibile collocare un pannello al margine sinistro di una scheda, impostare la sua proprietà *Alignment* a *alLeft*, quindi collocare uno splitter (anch'esso con allineamento *alLeft*) alla destra del pannello, e infine collocare un altro pannello (con allineamento *alLeft* oppure *alClient*) alla destra dello splitter.

Impostare la proprietà *MinSize* per specificare la minima dimensione lasciata dallo splitter quando si ridimensiona il suo controllo con esso confinante. Impostare la proprietà *Beveled* a **true** per far assumere al bordo dello splitter un aspetto tridimensionale.

Pulsanti e controlli simili

Menu a parte, i pulsanti rappresentano il modo più comune per attivare in un'applicazione un comando. C++Builder offre diversi controlli analoghi ai pulsanti:

Utilizzare questo componente:	Per fare ciò:
TButton	Presentare scelte di comando su pulsanti con solo testo
TBitBtn	Presentare scelte di comando su pulsanti con testo e simboli grafici
TSpeedButton	Creare pulsanti raggruppati in una barra strumenti
TCheckBox	Presentare opzioni di tipo on/off, sì/no
TRadioButton	Presentare una serie di scelte reciprocamente esclusive
TToolBar	Disporre pulsanti strumento ed altri controlli su più righe e regolarne automaticamente dimensioni e posizioni
TCoolBar	Visualizzare una raccolta di controlli per finestra, all'interno di bande spostabili e dimensionabili (solo VCL)

Controlli pulsante

L'utente fa clic con il mouse sui controlli pulsante per avviare azioni. I pulsanti hanno un'etichetta di testo che rappresenta l'azione. Il testo viene specificato assegnando un valore stringa alla proprietà *Caption*. La maggior parte dei pulsanti può essere selezionata anche premendo un tasto sulla tastiera, detto tasto di scelta rapida. La scorciatoia viene visualizzata sul pulsante come una lettera sottolineata.

I controlli pulsante sono utilizzati per consentire agli utenti di iniziare un'azione facendo clic su di essi. È possibile assegnare un'azione a un componente *TButton* creando per esso un gestore di evento *OnClick*. Facendo doppio clic su un pulsante in fase di progettazione, si passa nel gestore dell'evento *OnClick* del pulsante nell'editor di Codice.

- Impostare la proprietà *Cancel* a **true** se si desidera che il pulsante scateni il suo evento *OnClick* alla pressione del pulsante *Esc*.
- Impostare la proprietà *Default* a **true** se si desidera che il tasto *Invio* scateni l'evento *OnClick* del pulsante.

Pulsanti bitmap

Un pulsante bitmap (*BitBtn*) è un controllo pulsante che presenta sulla sua superficie un'immagine bitmap.

- Per scegliere una bitmap da assegnare al pulsante, impostare la proprietà *Glyph*.
- Utilizzare la proprietà *Kind* per configurare automaticamente un pulsante con un simbolo e un comportamento predefiniti.
- Per impostazione predefinita, il simbolo è posto alla sinistra di qualsiasi testo. Per spostarlo, utilizzare la proprietà *Layout*.
- Il simbolo e il testo vengono centrati automaticamente nel pulsante. Per spostarne la posizione, utilizzare la proprietà *Margin*. *Margin* determina il numero di pixel tra il bordo dell'immagine e il bordo del pulsante.
- Per impostazione predefinita, l'immagine e il testo sono separati da 4 pixel. Utilizzare la proprietà *Spacing* per aumentare o diminuire la distanza.
- I pulsanti bitmap possono avere 3 stati: su, giù e premuto. Impostare la proprietà *NumGlyphs* a 3 per mostrare una bitmap diversa per ciascuno stato.

Pulsanti di scelta rapida

I pulsanti di scelta rapida, che di solito hanno immagini sulla loro superficie, possono funzionare in gruppi. Di solito vengono utilizzati con pannelli per creare barre strumenti.

- Per fare sì che i pulsanti di scelta rapida si comportino come un gruppo, assegnare alla proprietà *GroupIndex* di tutti i pulsanti lo stesso valore diverso da zero.
- Per impostazione predefinita, i pulsanti di scelta rapida appaiono nello stato non selezionato (non premuto). Per visualizzare inizialmente un pulsante di scelta rapida come selezionato (premuto), impostare la proprietà *Down* a **true**.

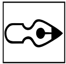
- Se la proprietà *AllowAllUp* è **true**, tutti i pulsanti di scelta rapida in un gruppo possono essere non selezionati. Impostare la proprietà *AllowAllUp* a **false** se si desidera che un gruppo di pulsanti agisca come un gruppo di pulsanti di opzione.

Per ulteriori informazioni su pulsanti di accesso rapido, fare riferimento alle sezioni “Aggiunta di una barra strumenti usando un componente panel” a pagina 8-47 e “Organizzazione delle azioni per barre e menu” a pagina 8-17.

Caselle di controllo

Una casella di controllo è un commutatore che permette all’utente di selezionare uno stato attivo/inattivo. Quando la scelta è attiva, la casella di controllo appare con un segno di spunta. In caso contrario, la casella di controllo è vuota. Le caselle di controllo vengono create utilizzando il componente *TCheckBox*.

- Impostare la proprietà *Checked* a **true** per far sì che la casella appaia selezionata per impostazione predefinita.
- Impostare la proprietà *AllowGrayed* a **true** per assegnare alla casella di controllo tre possibili stati: selezionato, deselezionato, e grigio.
- La proprietà *State* indica se la casella di controllo è selezionata (*cbChecked*), deselezionata (*cbUnchecked*), oppure grigia (*cbGrayed*).



I controlli casella di controllo visualizzano uno dei due possibili stati binari. Lo stato indeterminato viene utilizzato quando altre impostazioni rendono impossibile determinare il valore corrente della casella di controllo.

Pulsanti di opzione

I pulsanti di opzione presentano una serie di scelte reciprocamente esclusive. È possibile creare singoli pulsanti di opzione utilizzando *TRadioButton* oppure utilizzando il componente *radio group* (*TRadioGroup*) per organizzare automaticamente in gruppi più pulsanti di opzione. È possibile raggruppare i pulsanti di opzione in modo da consentire all’utente di effettuare una scelta da un insieme limitato di opzioni. Per maggiori informazioni, consultare la sezione “Controlli di raggruppamento” a pagina 9-14.

Un pulsante di opzione selezionato viene visualizzato come un cerchio con un pallino di riempimento. Se non è selezionato, il pulsante di opzione visualizza un cerchio vuoto. Per modificare lo stato visuale del pulsante di opzione, assegnare il valore **true** o **false** alla proprietà *Checked*.

Barre strumenti

Le barre strumenti rappresentano un modo semplice per disporre e gestire i controlli visuali. È possibile creare una barra strumenti a partire da un componente panel e da pulsanti di scelta rapida, oppure è possibile utilizzare il componente *ToolBar* e aggiungere alla barra strumenti una serie di pulsanti facendo clic destro e scegliendo il comando New Button.

Il componente *TToolBar* presenta diversi vantaggi: i pulsanti su una barra strumenti conservano automaticamente dimensioni e spaziatura uniformi; gli altri controlli conservano la propria posizione e altezza relative; i controlli possono essere disposti automaticamente su più righe quando lo spazio orizzontale non è sufficiente; inoltre, *TToolBar* consente di utilizzare varie opzioni di visualizzazione quali la trasparenza, i bordi popup oltre a spazi e divisori per raggruppare i controlli.

Grazie alle liste di azioni o alle bande di azioni è possibile utilizzare un set centralizzato di *azioni per le barre strumenti e per i menu*. Per i dettagli su come utilizzare liste di azioni con i pulsanti e le barre strumenti, consultare il paragrafo “Uso delle liste di azioni” a pagina 8-25.

Le barre strumenti possono anche fungere da genitore (contenere) di altri controlli come caselle di modifica, caselle combinate, e così via.

Cool bar (solo VCL)

Una coolbar contiene controlli figlio che possono essere spostati e ridimensionati in modo indipendente. Ciascun controllo risiede su un singolo nastro. L’utente posiziona i controlli trascinando la maniglia di dimensionamento alla sinistra di ciascun nastro.

Il componente coolbar richiede la presenza del file COMCTL32.DLL in versione 4.70 o successiva (solitamente il file è incluso nella directory Windows\System o Windows\System32) sia durante la progettazione sia in fase di esecuzione. Le coolbar non possono essere utilizzate in applicazioni multipiattaforma.

- La proprietà *Bands* include una raccolta di oggetti *TCoolBand*. In fase di progettazione, è possibile aggiungere, rimuovere o modificare i nastri con l’apposito editor. Per aprire il *Bands* editor, selezionare la proprietà *Bands* nell’Object Inspector, quindi fare doppio clic sulla colonna *Value* disposta alla destra, oppure fare clic sul pulsante ellissi (...) I nastri possono anche essere creati aggiungendo dalla tavolozza nuovi controlli a finestra.
- La proprietà *FixedOrder* determina se l’utente può riordinare i nastri.
- La proprietà *FixedSize* determina se i nastri hanno tutti un’altezza uniforme.

Controlli elenco

I componenti con elenchi presentano all’utente una serie di elementi fra cui effettuare una selezione. Diversi componenti visualizzano elenchi:

Utilizzare questo componente:	Per visualizzare:
TListBox	Un elenco di stringhe di testo
TCheckListBox	Un elenco con una casella di controllo prima di ciascun elemento
TComboBox	Una casella di testo a scorrimento con un elenco a discesa
TTreeView	Un elenco gerarchico

Utilizzare questo componente:	Per visualizzare:
TListView	Un elenco di elementi (trascinabili) con possibilità di visualizzare icone, colonne e intestazioni
	TIconView (solo CLX)
TIconView (CLX only)	Un elenco di elementi o dati su righe e colonne, visualizzati come icone piccole o grandi
TDateTimePicker	Una casella di riepilogo per immettere date oppure orari (solo VCL)
TMonthCalendar	Un calendario per selezionare date (solo VCL)

Utilizzare i componenti non visuali *TStringList* e *TImageList* per gestire serie di stringhe e immagini. Per maggiori informazioni sugli elenchi di stringhe, consultare "Operazioni con elenchi di stringhe" a pagina 4-16.

Caselle di riepilogo e caselle di riepilogo con spunta

Le caselle di riepilogo (*TListBox*) e le caselle di riepilogo con spunta visualizzano liste da cui gli utenti possono slezionare uno o più elementi.

- *Items* utilizza un oggetto *TStrings* per riempire il controllo con valori.
- *ItemIndex* indica qual è l'elemento selezionato nell'elenco.
- *MultiSelect* specifica se l'utente può selezionare più di un elemento per volta.
- *Sorted* determina se l'elenco è ordinato secondo l'ordine alfabetico.
- *Columns* specifica il numero di colonne nel controllo elenco.
- *IntegralHeight* specifica se la casella di riepilogo mostra solo quegli elementi che riempiono completamente lo spazio verticale (solo VCL).
- *ItemHeight* specifica l'altezza in pixel di ciascun elemento. La proprietà *Style* può far sì che la proprietà *ItemHeight* sia ignorata.
- La proprietà *Style* determina come un controllo elenco visualizza i propri elementi. Per impostazione predefinita, gli elementi sono visualizzati come stringhe. Modificando il valore di *Style*, è possibile creare caselle di riepilogo con *tracciamento personalizzato* in grado di visualizzare gli elementi in modo grafico o con altezze differenti. Per informazioni sui controlli con tracciamento personalizzato, consultare la sezione "Aggiunta di grafici ai controlli" a pagina 6-12.

Per creare una semplice casella di riepilogo:

1. All'interno del progetto, trascinare su una scheda un componente casella di riepilogo prelevandolo dalla Component palette.
2. Dimensionare la casella di riepilogo e impostarne l'allineamento.
3. Fare doppio clic alla destra della proprietà *Items* o fare clic sul pulsante ellissi per visualizzare lo String List Editor.
4. Utilizzare l'editor per immettere del testo organizzato su più righe per il contenuto della casella di riepilogo.
5. Scegliere OK.

Per consentire agli utenti di selezionare più elementi nella casella di riepilogo, è possibile utilizzare le proprietà *ExtendedSelect* e *MultiSelect*.

Caselle combinate

Una casella combinata (*TComboBox*) combina una casella di testo con un elenco a scorrimento. Quando l'utente immette dati nel controllo, scrivendo oppure selezionando uno dei valori dell'elenco, viene modificato il valore della proprietà *Text*. Se la proprietà *AutoComplete* è attiva, l'applicazione cerca nella lista e visualizza la corrispondenza più simile al dato via via immesso dall'utente.

Esistono tre tipi di caselle combinate: standard, a discesa (impostazione predefinita), e lista a discesa.

- Utilizzare la proprietà *Style* per selezionare il tipo di casella combinata necessaria.
- Utilizzare *csDropDown* se si vuole una casella di testo con un elenco a discesa. Utilizzare *csDropDownList* per rendere la casella di testo a sola lettura (costringendo così l'utente ad effettuare una scelta dall'elenco). Impostare la proprietà *DropDownCount* per modificare il numero di elementi visualizzato nell'elenco.
- Utilizzare *csSimple* per creare una casella combinata con un elenco fisso che non si chiude. Accertarsi di ridimensionare la casella combinata in modo da visualizzare gli elementi dell'elenco.
- Utilizzare *csOwnerDrawFixed* oppure *csOwnerDrawVariable* per creare caselle combinate con *tracciamento personalizzato* in grado di visualizzare graficamente gli elementi o con altezze differenti. Per informazioni sui controlli con tracciamento personalizzato, consultare la sezione "Aggiunta di grafici ai controlli" a pagina 6-12.

In esecuzione, le caselle combinate CLX funzionano in modo diverso rispetto alle caselle combinate della VCL. In CLX (ma non nella casella combinata della VCL), è possibile aggiungere un elemento a una casella combinata a discesa immettendo del testo nel campo di editing e premendo il tasto Invio. È possibile disattivare questa funzione impostando *InsertMode* a ciNone. Alla lista della casella combinata è anche possibile aggiungere elementi vuoti (nessuna stringa). Inoltre, se si mantiene premuto il tasto freccia, non ci si ferma sull'ultimo elemento dell'elenco della casella combinata. Si torna nuovamente all'inizio.

Viste ad albero

Un controllo tree view *(TTreeView)* visualizza gli elementi secondo una struttura con rientranze. Il controllo dispone di pulsanti che consentono di espandere e contrarre i nodi. Assieme alle etichette di testo degli elementi è possibile includere delle icone e visualizzare icone differenti per indicare se un nodo è espanso o contratto. È possibile anche includere elementi grafici, come, ad esempio, un segno di spunta, che è in grado di fornire informazioni sullo stato degli elementi.

- *Indent* imposta il numero di pixels che separano in orizzontale gli elementi dai rispettivi genitori.

- *ShowButtons* consente la visualizzazione di pulsanti '+' e '–' per indicare se un elemento può essere espanso.
- *ShowLines* consente la visualizzazione di linee di connessione per evidenziare le relazioni gerarchiche (solo VCL).
- *ShowRoot* determina se visualizzare o meno le linee che connettono gli elementi di livello superiore (solo VCL).

Per aggiungere elementi a un controllo vista ad albero in fase di progettazione, fare doppio clic sul controllo in modo da visualizzare l'editor TreeView *Items*. Gli elementi aggiunti diventano il valore della proprietà *Items*. È possibile modificare gli elementi in esecuzione utilizzando i metodi della proprietà *Items* , che è un oggetto di tipo *TTreeNodes*. *TTreeNodes* ha metodi per aggiungere, cancellare ed esaminare gli elementi nella vista albero.

Le viste ad albero possono visualizzare colonne e sottoelementi in modo analogo ai componenti listview in modalità vsReport.

Viste ad elenco

I controlli list view, creati utilizzando *TListView*, visualizzano elenchi in diversi formati. Utilizzare la proprietà *ViewStyle* per scegliere il tipo di elenco desiderato:

- *vsIcon* e *vsSmallIcon* visualizzano ciascun elemento come un'icona con un'etichetta. L'utente può trascinare gli elementi all'interno della finestra del controllo vista ad elenco (solo VCL).
- *vsList* visualizza gli elementi come icone con etichetta che non è possibile trascinare.
- *vsReport* visualizza gli elementi su righe separate con informazioni disposte in colonne. La colonna più a sinistra contiene una piccola icona e un'etichetta, mentre le colonne successive contengono i sottoelementi specificati dall'applicazione. Utilizzare la proprietà *ShowColumnHeaders* per visualizzare l'intestazione delle colonne.

Selettore data-ora e calendari mensili (solo VCL)

Il componente DateTimePicker visualizza una casella di riepilogo per immettere date oppure orari, mentre il componente MonthCalendar presenta un calendario per immettere date oppure intervalli di date. Per utilizzare questi componenti è necessario avere una versione 4.70 o la versione più recente del file COMCTL32.DLL (di solito presente nella directory Windows\System oppure Windows\System32) sia in fase di progettazione sia durante l'esecuzione. Non sono utilizzabili in applicazioni multipiattaforma.

Controlli di raggruppamento

Un'interfaccia grafica è più facile da utilizzare se controlli e informazioni correlate sono presentate in gruppi. C++Builder possiede diversi componenti per raggruppare i componenti:

Utilizzare questo componente:	Quando si desidera:
TGroupBox	Una casella di gruppo standard con un titolo
TRadioGroup	Un semplice gruppo di pulsanti di opzione
TPanel	Un gruppo di controlli più flessibile
TScrollBox	Una regione a scorrimento contenente controlli
TTabControl	Una serie di separatori di pagine reciprocamente esclusivi
TPageControl	Un insieme di separatori in stile notebook reciprocamente esclusivi, con pagine corrispondenti, ognune delle quali può contenere altri controlli
THeaderControl	Intestazioni di colonna dimensionabili

Caselle di gruppo e gruppi di pulsanti di opzione

Una casella di gruppo (*TGroupBox*) consente di disporre su una scheda più controlli correlati. I controlli raggruppati più di frequente sono i pulsanti di opzione. Dopo avere collocato su una scheda una casella di gruppo, selezionare i componenti dalla Component palette e collocarli nella casella di gruppo. La proprietà *Caption* contiene il testo che identifica la casella di gruppo in fase di esecuzione.

Il componente radio group (*TRadioGroup*) semplifica le operazioni necessarie per assemblare più pulsanti di opzione per farli operare insieme. Per aggiungere pulsanti di opzione a un componente radio group, modificare nell'Object Inspector la proprietà *Items*; ciascuna stringa in *Items* fa sì che nella casella di gruppo appaia un pulsante di opzione il cui titolo è rappresentato dalla stringa stessa. Il valore della proprietà *ItemIndex* determina qual è il pulsante di opzione attualmente selezionato. Impostando il valore della proprietà *Columns* è possibile visualizzare i pulsanti di opzione in una sola colonna oppure disposti su più colonne. Per modificare la spaziatura fra i pulsanti, dimensionare il componente radio group.

Pannelli

Il componente *TPanel* fornisce un contenitore generico per altri controlli. Di solito i pannelli sono utilizzati per raggruppare visivamente più componenti su una scheda. I pannelli possono essere allineati con la scheda in modo da conservare la stessa posizione relativa se la scheda fosse dimensionata. La proprietà *BorderWidth* determina l'ampiezza in pixels del bordo attorno a un pannello.

Su un pannello è possibile anche collocare altri controlli e utilizzare la proprietà *Align* per assicurare il corretto posizionamento sulla scheda di tutti i controlli nel gruppo. È

possibile impostare l'allineamento del pannello a alTop in modo che la sua posizione rimanga fissa anche se la scheda viene ridimensionata.

L'aspetto del pannello può essere modificato facendogli assumere un aspetto rialzato o incavato grazie alle proprietà *BevelOuter* e *BevelInner*. È possibile variare i valori di queste proprietà per creare differenti effetti 3-D visuali. È bene notare che se si desidera solamente uno smusso rialzato o incavato, è possibile utilizzare il controllo *TBevel* che utilizza in minor numero di risorse.

È possibile anche utilizzare uno o più pannelli per costruire barre di stato o aree per la visualizzazione di informazioni.

Caselle a scorrimento

Le caselle a scorrimento (*TScrollBox*) consentono di creare all'interno di una scheda delle aree con funzioni di scorrimento. Spesso le applicazioni devono visualizzare più informazioni di quante ce ne stiano in una determinata area. Alcuni controlli come le caselle di riepilogo, i campi memo e le stesse schede, sono in grado di far scorrere i propri contenuti.

Un altro utilizzo delle caselle a scorrimento consiste nel creare più aree a scorrimento (viste) in un'unica finestra.Le viste sono molto comuni nei programmi commerciali di elaborazione testi, nei fogli elettronici o nelle applicazioni per la gestione di progetti. Le caselle a scorrimento offrono maggior flessibilità in quanto consentono di definire in modo arbitrario aree di scorrimento all'interno di una scheda.

In modo analogo ai pannelli e alle caselle di gruppo, le caselle a scorrimento contengono altri controlli, come gli oggetti *TButton* e *TCheckBox*. Una casella a scorrimento di solito è invisibile. Se i controlli inseriti nella casella a scorrimento non sono contenuti completamente nell'area visibile, la casella a scorrimento visualizza automaticamente barre di scorrimento.

Un altro utilizzo di tali caselle è quello di restringere lo scorrimento in determinate aree della finestra, come ad esempio in una barra strumenti o in una barra di stato (componenti *TPanel*). Per evitare che si possa far scorrere una barra strumenti o una barra di stato, nascondere le barre di scorrimento e posizionare una casella a scorrimento nell'area del client della finestra, fra la barra strumenti e la barra di stato. Le barre di scorrimento associate alla casella a scorrimento sembreranno appartenere alla finestra, ma consentiranno solo di far scorrere l'area all'interno della casella.

Controlli separatore

Il componente tab (*TTabControl*) crea una serie di separatori che assomigliano ai divisori di un'agenda. È possibile creare separatori modificando nell'Object Inspector la proprietà Tab; ciascuna stringa in *Tabs* rappresenta un separatore. Il controllo tab è un singolo pannello contenente una serie di componenti. Per modificare l'aspetto del controllo si fa clic su di esso, è necessario scrivere un gestore dell'evento *OnChange*. Per creare una finestra di dialogo multipagina, utilizzare un controllo Page.

Controlli pagina

Il componente controllo pagina (*TPageControl*) consiste di un set di pagine adatto alla creazione di finestre di dialogo multipagina. Un controllo pagina visualizza più pagine sovrapposte, ognuna delle quali è un oggetto *TTabSheet*. Nell'interfaccia utente, una pagina viene selezionata facendo clic sul separatore di pagina nella parte superiore del controllo.

Per creare una nuova pagina in un controllo pagina in fase di progettazione, fare clic destro sul controllo e scegliere il comando New page. In esecuzione, è possibile aggiungere nuove pagine creando l'oggetto per la pagina e impostandone la proprietà *PageControl*:

```
TTabSheet *pTabSheet = new TTabSheet(PageControl1);
pTabSheet->PageControl = PageControl1;
```

Per accedere alla pagina attiva utilizzare la proprietà *ActivePage*. Per modificare la pagina attiva, è possibile impostare le proprietà *ActivePage* o *ActivePageIndex*.

Controlli intestazione

Un controllo header (*THeaderControl*) è una serie di intestazioni di colonna che l'utente può selezionare o ridimensionare in fase di esecuzione. Modificare la proprietà *Sections* del controllo per aggiungere o modificare le intestazioni. È possibile collocare sezioni di intestazione su colonne o campi. Ad esempio, le sezioni di intestazione potrebbero essere collocate sulle caselle di riepilogo (*TListBox*).

Controlli di visualizzazione

Esistono diversi modi per fornire agli utenti le informazioni sullo stato di un'applicazione. Ad esempio, alcuni componenti, incluso *TForm*, hanno una proprietà *Caption* che può essere impostata durante l'esecuzione. È anche possibile creare finestre di dialogo per visualizzare messaggi. Inoltre, i componenti elencati di seguito risultano particolarmente utili per fornire in esecuzione un feedback visivo.

Utilizzare il componente o la proprietà:	Se si vuole:
TStatusBar	Visualizzare un'area di stato (di solito nella parte inferiore di una finestra)
TProgressBar	Mostrare lo stato di avanzamento di una determinata operazione
Hint e *ShowHint*	Attivare il fumetto di aiuto o i suggerimenti
HelpContext e *HelpFile*	Collegare la Guida in linea contestuale

Barre di stato

Sebbene sia possibile utilizzare un pannello per creare una barra di stato, è più semplice utilizzare il componente *TStatusBar*. Per impostazione predefinita, la

proprietà *Align* della barra di stato è impostata ad *alBottom*, che ne determina automaticamente posizione e dimensione.

Se è necessario visualizzare nella barra di stato solo una stringa di testo per volta, impostare la proprietà *SimplePanel* a **true** e utilizzare la proprietà *SimpleText* per controllare il testo visualizzato nella barra di stato.

È possibile che una barra di stato venga suddivisa in varie aree di testo, chiamate pannelli. Per creare i pannelli, modificare nell'Object Inspector la proprietà *Panels* impostando per ciascun pannello, mediante il Panel editor, le proprietà *Width*, *Alignment*, e *Text*. La proprietà *Text* di ogni pannello contiene il testo visualizzato nel pannello.

Barre di avanzamento

Se l'applicazione deve compiere un'operazione molto lunga, è possibile utilizzare una barra di avanzamento per mostrare lo stato di avanzamento dell'operazione. Una barra di avanzamento visualizza una riga punteggiata che cresce da sinistra a destra.

Figura 9.2 Una barra di avanzamento



La proprietà *Position* tiene traccia della lunghezza della linea punteggiata. Le proprietà *Max* e *Min* determinano l'intervallo di validità di *Position*. Per fare sì che la riga cresca, incrementare *Position* chiamando i metodi *StepBy* o *StepIt*. La proprietà *Step* determina l'incremento utilizzato da *StepIt*.

Proprietà Help e Hint

Quasi tutti i controlli visuali possono visualizzare in fase di esecuzione sia una Guida contestuale sia dei suggerimenti. Le proprietà *HelpContext* e *HelpFile* sono usate per determinare un numero di contesto per la Guida e il nome del file della Guida del controllo.

La proprietà *Hint* contiene la stringa di testo che appare quando l'utente sposta il puntatore del mouse su un controllo o su un elemento di menu. Per attivare i suggerimenti, impostare *ShowHint* a **true**; se si imposta *ParentShowHint* a **true**, la proprietà *ShowHint* del controllo assumerà lo stesso valore della proprietà del genitore.

Griglie

Le griglie visualizzano informazioni disposte su righe e colonne. Se si sta scrivendo un'applicazione database, utilizzare i componenti *TDBGrid* o *TDBCtrlGrid* descritti nel Capitolo 19, "Uso dei controlli dati". Oppure, utilizzare una griglia draw standard o una griglia stringa.

Griglie grafiche

Una griglia grafica (*TDrawGrid*) visualizza dati arbitrari in formato tabellare. Per riempire le celle della griglia si deve scrivere un gestore per l'evento *OnDrawCell*.

- Il metodo *CellRect* restituisce le coordinate video di una cella specificata, mentre il metodo *MouseToCell* restituisce la colonna e la riga della cella situata alle coordinate video specificate. La proprietà *Selection* indica i bordi delle celle attualmente selezionate.
- La proprietà *TopRow* determina quale riga è attualmente nella parte superiore della griglia. La proprietà *LeftCol* determina la prima colonna visibile sulla sinistra. *VisibleColCount* e *VisibleRowCount* rappresentano il numero di colonne e righe visibili nella griglia.
- È possibile modificare l'ampiezza o altezza di una colonna o di una riga mediante le proprietà *ColWidths* e *RowHeights*.Impostare lo spessore delle linee della griglia mediante la proprietà *GridLineWidth*. Aggiungere alla griglia le barre di scorrimento mediante la proprietà *ScrollBars*.
- Mediante le proprietà *FixedCols* e *FixedRows* è possibile decidere se avere colonne o righe fisse non soggette a scorrimento. Alle colonne e alle righe fisse è possibile assegnare un colore usando la proprietà *FixedColor*.
- Le proprietà *Options*, *DefaultColWidth* e *DefaultRowHeight* influenzano anche l'aspetto e il comportamento della griglia.

Griglie per stringhe

Il componente grid string è un discendente di *TDrawGrid* che aggiunge funzionalità specializzate per semplificare la visualizzazione di stringhe. La proprietà *Cells* elenca le stringhe per ciascuna cella della griglia; la proprietà *Objects* elenca gli oggetti associati a ciascuna stringa. Tramite le proprietà *Cols* e *Rows* è possibile accedere a tutte le stringhe e gli oggetti associati ad una particolare colonna o riga.

Editor di liste di valori (solo VCL)

TValueListEditor è una griglia specializzata per editare liste di stringhe contenenti coppie nome/valore nel formato Name=*Value*. I nomi e i valori sono memorizzati come discendenti di *TStrings* che è il valore della proprietà *Strings*. È possibile cercare il valore di un qualsiasi nome utilizzando la proprietà *Values*. *TValueListEditor* non è utilizzabile per programmi multipiattaforma.

La griglia contiene due colonne, una per i nomi e una per i valori. Per impostazione predefinita, la colonna Name è detta "Key" e la colonna *Value* è detta "*Value*". È possibile modificare queste impostazioni predefinite impostando la proprietà *TitleCaptions*. È possibile omettere questi titoli utilizzando la proprietà *DisplayOptions* (che controlla anche il dimensionamento quando si ridimensiona il controllo.)

Utilizzando la proprietà *KeyOptions* è possibile controllare se gli utenti possono modificare la colonna Name. *KeyOptions* contiene opzioni separate per consentire l'editing, l'aggiunta di nuovi nomi, la cancellazione di nomi e controllare se i nuovi nomi devono essere univoci.

Utilizzando la proprietà *ItemProps* è possibile controllare le modalità secondo cui gli utenti editano le voci nella colonna *Value*. Ogni elemento ha un oggetto separato *TItemProp* che consente di

- Fornire una maschera di immissione per controllare la validità dei dati immessi.
- Specificare la lunghezza massima per i valori.
- Contrassegnare il valore come a sola lettura.
- Specificare che l'editor della lista di valori visualizzi una freccia verso il basso per aprire una lista di possibili valori tra cui effettuare una scelta oppure un pulsante ellissi che inneschi un evento utilizzabile per visualizzare una finestra di dialogo in cui immettere i valori.

  Se si specifica che c'è una freccia verso il basso, è necessario fornire la lista di valori da cui effettuare la scelta. Questa lista può essere una lista statica (la proprietà *PickList* dell'oggetto *TItemProp*) oppure è possibile aggiungere dinamicamente i valori in fase di esecuzione utilizzando l'evento *OnGetPickList* dell'editor della lista di valori. È possibile anche combinare queste modalità e avere una lista statica modificabile con il gestore di evento *OnGetPickList*.

  Se si specifica che c'è un pulsante ellissi, è necessario fornire la risposta che si verifica quando l'utente fa clic su quel pulsante (inclusa l'impostazione di un valore, se è il caso). Si fornisce questa risposta scrivendo un gestore di evento *OnEditButtonClick*.

Controlli grafici

I componenti seguenti semplificano l'incorporazione di elementi grafici in un'applicazione.

Utilizzare questo componente:	**Per visualizzare:**
TImage	File grafici
TShape	Forme geometriche
TBevel	Linee 3D e frame
TPaintBox	Grafici disegnati dal proprio programma in fase di esecuzione
TAnimate	File AVI (solo VCL)

Si noti che questi includono le comuni routine di disegno (*Repaint*, *Invalidate* e così via) che non avranno mai bisogno di ricevere il fuoco.

Immagini

Il componente image (immagine) visualizza un'immagine grafica, come una bitmap, un'icona o un metafile. La proprietà *Picture* determina l'immagine grafica da visualizzare. Utilizzare *Center*, *AutoSize*, *Stretch* e *Transparent* per impostare le opzioni di visualizzazione. Per maggiori informazioni consultare "Panoramica sulla programmazione dei grafici" a pagina 10-1.

Forme

Il componente shape (forma) visualizza una forma geometrica.È un controllo che è basato su finestra (non basato su widget in CLX) e pertanto non può ricevere l'input dell'utente. La proprietà *Shape* determina la forma assunta dal controllo.Per modificare il colore della forma o aggiungere un motivo di riempimento, utilizzare la proprietà *Brush* che contiene un oggetto *TBrush*. Il modo in cui la forma viene tracciata dipende dalle proprietà *Color* e *Style* di *TBrush*.

Linee smussate

Il componente bevel (*TBevel*) è una linea che può apparire incavata oppure a sbalzo. Alcuni componenti, come *TPanel*, hanno proprietà intrinseche per creare bordi smussati. Quando tali proprietà non sono disponibili, l'uso di *TBevel* consente di creare contorni smussati, riquadri o cornici.

Caselle di disegno

Il componente paint box (*TPaintBox*) consente all'applicazione di tracciare un disegno su una scheda. Per riprodurre direttamente un'immagine sul *Canvas* della casella di disegno occorre scrivere un gestore per l'evento *OnPaint*. Non è consentito disegnare al di fuori dei bordi della casella di disegno. Per maggiori informazioni consultare "Panoramica sulla programmazione dei grafici" a pagina 10-1.

Controlli Animation (solo VCL)

Il componente animation (animazione) è una finestra che visualizza senza sonoro un file Audio Video Interleaved (AVI). Un file AVI è composto da una serie di frame in formato bitmap, simile ad un film. Sebbene i file AVI possano contenere una traccia sonora, i controlli animazione funzionano soltanto con filmati AVI senza suono. I file utilizzati possono essere sia in formato AVI non compresso sia in formato AVI compresso mediante la codifica RLE (run-lenght encoding). Il controllo di animazione non può essere utilizzato in programmi multipiattaforma.

Di seguito sono elencate alcune importanti proprietà di un controllo animation:

- *ResHandle* è l'handle di Windows per il modulo che contiene il filmati AVI sotto forma di risorsa. Impostare in fase di esecuzione *ResHandle* allo handle dell'istanza

oppure allo handle di modulo di quel modulo che include la risorsa di animazione. Dopo aver impostato *ResHandle*, impostare la proprietà *ResID* o *ResName* per specificare quale risorsa nel modulo indicato è il filmato AVI che dovrebbe essere visualizzato dal controllo animazione.

- Impostare *AutoSize* a **true** per fare in modo che il controllo animazione regoli le proprie dimensioni alle dimensione dei fotogrammi nel filmato AVI.
- *StartFrame* e *StopFrame* specificano a quali fotogrammi iniziare e fermare il filmato.
- Impostare *CommonAVI* per visualizzare uno dei filmati AVI di Windows forniti con Shell32.DLL.
- Specificare quando iniziare e interrompere l'animazione impostando, rispettivamente, la proprietà *Active* a **true** o **false**, e quante ripetizioni eseguire impostando la proprietà *Repetitions*.
- La proprietà *Timers* permette di visualizzare i frame utilizzando un timer. Questo è utile per sincronizzare la sequenza di animazioni con altre azioni, come ad esempio l'esecuzione di una traccia audio.

Capitolo

10

Operazioni con elementi grafici e multimediali

L'uso di elementi grafici e multimediali può aggiungere un tocco di raffinatezza alle proprie applicazioni. C++Builder offre vari modi per introdurre queste funzionalità nelle applicazioni. Per aggiungere elementi grafici, è possibile inserire immagini disegnate in precedenza in fase di progettazione, è possibile crearle usando un controllo grafico in fase di esecuzione oppure tracciarle mediante un controllo grafico in fase di esecuzione. Per aggiungere funzionalità multimediali, C++Builder include uno speciale componente in grado di riprodurre file audio e video.

I componenti multimediali sono disponibili solo nella VCL.

Panoramica sulla programmazione dei grafici

I componenti grafici della VCL, definiti nella unit Graphics, incapsulano la Graphics Device Interface (GDI) di Windows, semplificando notevolmente l'aggiunta di grafici alle applicazioni Windows. I componenti grafici CLX definiti nella unit QGraphics incapsulano i widget grafici Qt per l'aggiunta della grafica alle applicazioni multipiattaforma.

Per disegnare grafici in un'applicazione C++Builder, si deve disegnare sul *canvas* di un oggetto, anziché direttamente sull'oggetto stesso. Il canvas è una proprietà dell'oggetto, ed è essa stessa un oggetto. Il principale vantaggio dell'oggetto Canvas è che gestisce efficacemente le risorse e si occupa del contesto del dispositivo, cosicché i programmi possono usare gli stessi metodi, indipendentemente dal fatto che si disegni sullo schermo, su una stampante, sui bitmap o sui metafiles (disegno in CLX). I Canvas sono disponibili solo in fase di esecuzione, pertanto tutte le operazioni sui canvas devono essere gestite mediante del codice apposito.

Poiché *TCanvas* è un gestore di risorse wrapper nel contesto dei dispositivi di Windows, nel canvas si possono usare anche tutte le funzioni GDI di Windows. La proprietà *Handle* del canvas è lo Handle di contesto del dispositivo.

TCanvas è un gestore di risorse wrapper attorno a un painter di Qt. La proprietà *Handle* del canvas èun puntatore con tipo a un'istanza dell'oggetto painter di QT. Il fatto che il puntatore all'istanza sia esposto permette di utilizzare funzioni grafiche della libreria Qt a basso livello che richiedono QPainterH.

La modalità di visualizzazione delle immagini grafiche nelle applicazioni dipende dal tipo di oggetto sul cui canvas si sta disegnando. Se si sta disegnando direttamente sul canvas di un controllo, l'immagine viene visualizzata immediatamente. Tuttavia, se si sta disegnando su un'immagine fuori schermo, come ad esempio un canvas di un oggetto *TBitmap*, l'immagine non viene visualizzata finché un controllo non la copia dalla bitmap sul canvas di un altro controllo. Ciò significa che, quando si disegnano delle bitmap e le si assegna a un controllo image, l'immagine appare solo quando il controllo ha l'opportunità di elaborarne il messaggio (VCL) o l'evento (CLX) *OnPaint*).

Durante le operazioni con elementi grafici, spesso si incontrano termini quali *tracciamento* e *disegno*:

- Il tracciamento è la creazione di un singolo e determinato elemento grafico, come una linea o una figura, mediante del codice. Nel codice, si impartisce a un oggetto il comando di tracciare un determinato elemento grafico in una certa posizione del suo canvas mediante una chiamata al metodo di tracciamento del canvas.
- Il disegno è la creazione dell'aspetto globale di un oggetto. Il disegno di solito comporta il tracciamento. Ciò significa che, in risposta agli eventi *OnPaint*, un oggetto di solito traccia alcuni elementi grafici. Una casella di testo, ad esempio, disegna se stessa tracciando dapprima un rettangolo e successivamente del testo al suo interno. Un controllo shape, d'altro canto, disegna se stesso tracciando un singolo elemento grafico.

L'esempio all'inizio del capitolo illustra come tracciare vari elementi grafici, ma questi lo fanno in risposta ad eventi *OnPaint*. Le sezioni successive mostrano come compiere le stesse operazioni di tracciamento in risposta ad altri eventi.

Aggiornamento dello schermo

Talvolta il sistema operativo determina che gli oggetti sullo schermo hanno bisogno di rinnovare il loro aspetto, e quindi genera in Windows dei messaggi WM_PAINT, che la VCL inoltra agli eventi *OnPaint* . (Se si utilizza CLX per lo sviluppo multipiattaforma, viene generato un evento Paint, che CLX indirizza a eventi *OnPaint*). Se si è scritto un gestore di evento *OnPaint* per quell'oggetto, esso verrà chiamato quando si userà il metodo *Refresh*. Il nome predefinito generato per il gestore di evento *OnPaint* in una scheda è *FormPaint*. A volte, può essere necessario usare il metodo *Refresh* per effettuare l'aggiornamento di un componente o di una scheda. Per esempio, si potrebbe chiamare *Refresh* nel gestore di evento *OnResize* della scheda per visualizzare di nuovo tutti gli elementi grafici oppure, nel caso si usi la VCL, per dipingere lo sfondo di una scheda.

Mentre alcuni sistemi operativi gestiscono automaticamente il ridisegno dell'area client di una finestra che è stata invalidata, Windows non lo fa. Nel sistema operativo Windows tutto ciò che viene disegnato sullo schermo è permanente. Quando una scheda o un controllo viene oscurato temporaneamente, per esempio durante il disegno di una finestra, la scheda o il controllo devono dipingere di nuovo l'area oscurata quando vengono riesposti. Per ulteriori informazioni sul messaggio WM_PAINT, consultare la Guida in linea di Windows.

Se si usa il controllo *TImage* per visualizzare sulla scheda un'immagine grafica, le operazioni relative alla pittura e all'aggiornamento del grafico contenuto in *TImage* vengono gestite automaticamente. La proprietà *Picture* specifica la bitmap attuale, il disegno o un altro oggetto grafico visualizzato da *TImage*. È possibile anche impostare la proprietà *Proportional* per garantire che l'immagine possa essere visualizzata completamente nel controllo immagine senza alcuna distorsione. Disegnando su un controllo *TImage*, si crea un'immagine persistente. Conseguentemente, non occorre fare nulla per ridisegnare l'immagine contenuta. Al contrario, il canvas di *TPaintBox* mappa direttamente sul dispositivo di schermo (VCL) o sul painter (CLX), e pertanto qualsiasi cosa disegnata sul canvas di *PaintBox* è transitoria. Ciò è vero per quasi tutti i controlli, compresa la scheda stessa. Perciò, se si disegna o si dipinge su un *TPaintBox* nel suo costruttore, occorrerà aggiungere quel codice al gestore dell'evento *OnPaint* affinché l'immagine venga ridipinta ogni volta che l'area client viene invalidata.

Tipi di oggetti grafici

Le librerie VCL/CLX forniscono gli oggetti grafici elencati nella Tabella 10.1. Questi oggetti hanno metodi per disegnare sul canvas, come descritto nel paragrafo "Uso dei metodi Canvas per disegnare oggetti grafici" a pagina 10-10 e per caricare e salvare file grafici, come descritto nel paragrafo "Caricamento e salvataggio dei file grafici" a pagina 10-20.

Tabella 10.1 Tipi di oggetti grafici

Oggetto	Descrizione
Picture	Serve per contenere un'immagine grafica. Per aggiungere altri formati di file grafici, si deve usare il metodo *Register* di Picture. Questo metodo consente di gestire file arbitrari come la visualizzazione delle immagini nel relativo controllo.
Bitmap	Un potente oggetto grafico usato per creare, manipolare (ridurre in scala, far scorrere, ruotare e dipingere) e memorizzare immagini su disco. La creazione di copie di una bitmap è molto rapida, poiché viene copiato l'*handle*, non l'immagine.
Clipboard	Rappresenta il contenitore per qualsiasi testo o grafico da tagliare, copiare o incollare da o in un'applicazione. Con la clipboard è possibile ottenere e recuperare dati secondo il formato appropriato; gestire il conteggio dei riferimenti, e l'apertura e la chiusura dela clipboard; gestire e manipolare i formati per gli oggetti nella clipboard.

Tabella 10.1 Tipi di oggetti grafici (continua)

Oggetto	Descrizione
Icon	Rappresenta il valore caricato da un file icona di Windows (file ::ICO).
Metafile (VCL only) Drawing (CLX only)	Contiene un file, che registra le operazioni necessarie per costruire un'immagine, anziché contenere gli effettivi pixel della bitmap dell'immagine. I metafile o i disegni sono estremamente scalabili senza perdita dei dettagli dell'immagine e spesso richiedono molta meno memoria delle bitmap, in modo particolare per i dispositivi ad alta risoluzione, come le stampanti. Comunque, i metafile e i disegni non vengono visualizzati velocemente quanto le bitmap. Usare un metafile o un disegno quando la versatilità o la precisione sono più importanti delle prestazioni.

Proprietà e metodi comuni dell'oggetto Canvas

La Tabella 10.2 elenca le proprietà dell'oggetto Canvas usate più di frequente. Per un elenco completo delle proprietà e dei metodi, consultare il componente *TCanvas* nella Guida in linea.

Tabella 10.2 Proprietà comuni dell'oggetto Canvas

Proprietà	Descrizione
Font	Specifica il font da usare quando si scrive del testo nell'immagine. Per specificare il font, il colore, le dimensioni e lo stile del font, si devono impostare le proprietà dell'oggetto TFont.
Brush	Determina il colore e il modello che il canvas usa per riempire i grafici e gli sfondi. Per specificare il colore e il modello o la bitmap da usare quando si riempiono gli spazi sul canvas, si devono impostare le proprietà dell'oggetto TBrush.
Pen	Specifica il tipo di penna che il canvas usa per tracciare le linee e disegnare le figure. Per specificare il colore, lo stile, la larghezza e il modo della penna, si devono impostare le proprietà dell'oggetto TPen.
PenPos	Specifica la posizione di disegno attiva della penna.
Pixels	Specifica il colore dell'area di pixel all'interno del ClipRect attivo.

Queste proprietà sono descritte in modo più approfondito nella sezione "Uso delle proprietà dell'oggetto Canvas" a pagina 10-5.

La Tabella 10.3 riporta un elenco dei vari metodi che è possibile usare:

Tabella 10.3 Metodi comuni dell'oggetto Canvas

Metodo	Descrizione
Arc	Disegna un arco sull'immagine lungo il perimetro dell'ellisse delimitata dal rettangolo specificato.
Chord	Disegna una figura chiusa rappresentata dall'intersezione di una linea e di un'ellisse.
CopyRect	Copia parte di un'immagine da un altro canvas nel canvas.
Draw	Rappresenta l'oggetto grafico specificato dal parametro Graphic sul canvas nella posizione indicata dalle coordinate (*X*, *Y*).

Tabella 10.3 Metodi comuni dell'oggetto Canvas (continua)

Metodo	Descrizione
Ellipse	Disegna l'ellisse definita da un rettangolo di delimitazione sul canvas.
FillRect	Riempie il rettangolo specificato sul canvas usando il pennello attivo.
FloodFill (VCL only)	Riempie un'area del canvas usando il pennello attivo.
FrameRect	Disegna un rettangolo usando il pennello del canvas per disegnare il bordo.
LineTo	Disegna una linea sul canvas dalla posizione della PenPos al punto specificato da *X* e *Y*, e imposta la posizione della penna a (*X*, *Y*).
MoveTo	Cambia la posizione di disegno attiva al punto (*X*,*Y*).
Pie	Disegna a forma di torta la sezione dell'ellisse delimitata dal rettangolo (X1, Y1) e (X2, Y2) sul canvas.
Polygon	Disegna una serie di linee sul canvas collegando i punti per cui passa e chiudendo la figura con una linea dall'ultimo punto al primo.
Polyline	Disegna una serie di linee sul canvas con la penna attiva, collegando ciascuno dei punti passati in Points.
Rectangle	Traccia sul canvas un rettangolo il cui angolo superiore sinistro è posizionato nel punto (X1, Y1) e il cui angolo inferiore destro è posizionato nel punto (X2, Y2). Utilizzare il metodo *Rectangle* per tracciare un riquadro con Pen e riempilo con Brush.
RoundRect	Disegna a rettangolo con gli angoli arrotondati sul canvas.
StretchDraw	Disegna un grafico sul canvas in modo che l'immagine trovi posto nel rettangolo specificato. Il grafico può aver bisogno di cambiare le dimensioni o l'aspetto per stare nel rettangolo.
TextHeight, TextWidth	Restituisce, rispettivamente, l'altezza e la larghezza di una stringa nel font corrente. L'altezza include lo spazio tra le righe.
TextOut	Scrive una stringa sul canvas, partendo dal punto (*X*,*Y*), e quindi aggiorna la posizione della penna PenPos alla fine della stringa.
TextRect	Scrive una stringa dentro una regione; qualsiasi parte della stringa che cade al di fuori della regione non viene visualizzata.

Questi metodi sono descritti in modo più approfondito nel paragrafo "Uso dei metodi Canvas per disegnare oggetti grafici" a pagina 10-10.

Uso delle proprietà dell'oggetto Canvas

Con l'oggetto Canvas è possibile impostare le proprietà di una penna per disegnare linee, di un pennello per riempire le figure, di un font per scrivere il testo e di un array di pixel per rappresenta l'immagine.

In questa sezione vengono trattati i seguenti argomenti:

- Uso delle penne.
- Uso dei pennelli.
- Lettura e impostazione dei pixel.

Uso delle penne

La proprietà *Pen* di un canvas controlla il modo in cui appaiono le linee, comprese quelle disegnate come contorno delle figure. Disegnare una retta significa in realtà cambiare un gruppo di pixel che si trovano tra due punti.

La penna stessa ha quattro proprietà che possono essere modificate: *Color*, *Width*, *Style* e *Mode*.

- La proprietà *Color* modifica il colore della penna.
- La proprietà *Width* modifica la larghezza della penna.
- La proprietà *Style* modifica lo stile della penna.
- La proprietà *Mode* cambia il modo della penna.

I valori di queste proprietà determinano come la penna cambia i pixel della linea. Per impostazione predefinita, ogni penna comincia con il colore nero, la larghezza di 1 pixel, lo stile continuo e il modo chiamato copy che sovrascrive tutto ciò che si trova già sul canvas.

È possibile utilizzare *TPenRecall* per salvare e ripristinare rapidamente le proprietà delle penne.

Modifica del colore della penna

È possibile impostare il color di una penna come si fa con qualsiasi altra proprietà *Color* in fase di esecuzione. Il colore di una penna determina il colore delle linee che la penna disegna, comprese quelle disegnate come contorni delle figure, oltre che le altre linee e le spezzate. Per cambiare il colore della penna, si deve assegnare un valore alla proprietà *Color* della penna.

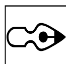
Per consentire all'utente di scegliere un nuovo colore per la penna, occorre mettere una griglia di colori nella barra strumenti della penna. Questa griglia può impostare i colori del primo piano e dello sfondo. Per uno stile di penna non griglia, occorre considerare il colore dello sfondo, che viene disegnato negli spazi tra i segmenti. Il colore dello sfondo viene impostato nella proprietà *Color* del pennello.

Poiché l'utente sceglie un nuovo colore facendo clic sulla griglia, il codice seguente cambia il colore della penna in risposta all'evento *OnClick*:

```
void __fastcall TForm1::PenColorClick(TObject *Sender)
{
  Canvas->Pen->Color = PenColor->ForegroundColor;
}
```

Modifica della larghezza della penna

La larghezza di una penna determina lo spessore, in pixel, delle linee che vengono disegnate.



Se lo spessore è maggiore di 1, Windows disegna sempre linee continue, indipendentemente dal valore della proprietà *Style* della penna.

Per modificare la larghezza della penna, si deve assegnare un valore numerico alla proprietà *Width* della penna.

Si supponga di avere una barra di scorrimento sulla barra strumenti della penna per impostare i valori dello spessore della penna. Si supponga inoltre di voler aggiornare l'etichetta accanto alla barra di scorrimento per fornire informazioni di controllo all'utente. Usando la posizione della barra di scorrimento per determinare la larghezza della penna, questa viene aggiornata ogni volta che la posizione cambia.

Il codice che segue indica come gestire l'evento *OnChange* della barra di scorrimento:

```
void __fastcall TForm1::PenWidthChange(TObject *Sender)
{
  Canvas->Pen->Width = PenWidth->Position;         // set the pen width directly
  PenSize->Caption = IntToStr(PenWidth->Position); // convert to string
}
```

Modifica dello stile della penna

La proprietà *Style* della penna consente di impostare linee continue, linee tratteggiate, linee punteggiate, e così via.

Se si sta sviluppando un'applicazione multipiattaforma da distribuire in ambiente Windows, Windows non supporta gli stili tratteggiato e punteggiato per penne più spesse di un pixel e disegna come continue le penne più spesse, indipendentemente dallo stile specificato.

L'impostazione delle proprietà di una penna è ideale per avere controlli differenti che condividono lo stesso gestore per gestire gli eventi. Per determinare quale controllo ha di fatto ricevuto l'evento, si deve contrassegnare il parametro *Sender*.

Per creare un gestore per gli eventi *OnClick* relativi ad un gruppo di sei pulsanti di stile su una barra strumenti della penna, occorre:

1 Selezionare tutti e sei i pulsanti e quindi Object Inspector | Events | *OnClick* e nella colonna Handler scrivere `SetPenStyle`.

   C++Builder genera un gestore di evento vuoto di nome *SetPenStyle* e lo collega agli eventi *OnClick* di tutti e sei pulsanti.

2 Riempire il gestore di evento generato impostando lo stile della penna in funzione del valore di *Sender*, ossia del controllo che ha inviato l'evento *OnClick*:

```
void __fastcall TForm1::SetPenStyle(TObject *Sender)
{
  if (Sender == SolidPen)
    Canvas->Pen->Style = psSolid;
  else if (Sender == DashPen)
    Canvas->Pen->Style = psDash;
  else if (Sender == DotPen)
    Canvas->Pen->Style = psDot;
  else if (Sender == DashDotPen)
    Canvas->Pen->Style = psDashDot;
  else if (Sender == DashDotDotPen)
    Canvas->Pen->Style = psDashDotDot;
'  else if (Sender == ClearPen)
    Canvas->Pen->Style = psClear;
}
```

Il gestore di evento visto in precedenza potebbe essere ulteriormente ridotto ponendo le costanti relative allo stile della penna nelle proprietà *Tag* dei pulsanti relativi allo stile delle penne. In questo caso, il codice dell'evento sarebbe simile al seguente::

```
void __fastcall TForm1::SetPenStyle(TObject *Sender)
{
  if (Sender->InheritsFrom (__classid(TSpeedButton))
    Canvas->Pen->Style = (TPenStyle) ((TSpeedButton *)Sender)->Tag;
}
```

Modifica del modo della penna

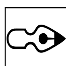
La proprietà *Mode* della penna permette di specificare i vari modi per combinare il colore della penna con il colore sul canvas. Per esempio, la penna può essere sempre nera, essere l'inverso del colore dello sfondo del canvas, l'inverso del colore della penna, e così via. Per informazioni dettagliate, consultare *TPen* nella Guida in linea.

Rilevazione della posizione della penna

La posizione corrente di disegno, cioè la posizione dalla quale la penna inizia a disegnare la successiva linea, è chiamata posizione della penna. Il canvas memorizza questa posizione nella proprietà *PenPos*.La posizione della penna influenza solo il disegno delle linee; per le figure e per il testo, occorre specificare le coordinate necessarie.

Per impostare la posizione della penna, si deve chiamare il metodo *MoveTo* del canvas. Per esempio, il codice seguente sposta la posizione della penna nell'angolo superiore sinistro del canvas:

```
Canvas->MoveTo(0, 0);
```



Disegnando una linea con il metodo *LineTo*, si sposta anche la posizione attuale al punto finale della linea.

Uso dei pennelli

La proprietà *Brush* di un canvas controlla il modo in cui le aree vengono riempite, compreso l'interno delle figure. Riempire un'area con un pennello equivale a cambiare una grande quantità di pixel adiacenti in un modo specificato.

Il pennello ha tre proprietà che è possibile manipolare:

- Proprietà *Color*: modifica il colore di riempimento.
- La proprietà *Style* modifica lo stile del pennello.
- La proprietà *Bitmap* usa una bitmap come modello di pennello.

I valori di queste proprietà determinano il modo in cui il canvas riempie le figure o le altre aree. Per impostazione predefinita, ogni pennello comincia con il colore bianco, con uno stile continuo e con nessuna bitmap come modello.

È possibile utilizzare *TBrushRecall* per salvare e ripristinare rapidamente le proprietà dei pennelli.

Modifica del colore del pennello

Il colore di un pennello determina quale colore usa il canvas per riempire le figure. Per cambiare il colore di riempimento, occorre assegnare un valore alla proprietà *Color* del pennello. Il pennello viene usato per il colore dello sfondo nel testo e del disegno della linea, cosicché, normalmente, basta impostare la proprietà *Color* dello sfondo.

È possibile impostare il colore del pennello proprio come si fa con il colore della penna, in risposta ad un clic su una griglia di colori sulla barra strumenti del pennello (consultare "Modifica del colore della penna" a pagina 10-6):

```
void __fastcall TForm1::BrushColorClick(TObject *Sender)
{
  Canvas->Brush->Color = BrushColor->BackgroundColor;
}
```

Modifica dello stile del pennello

Lo stile di un pennello determina quale modello usa il canvas per riempire le figure. È possibile specificare diversi modi per combinare il colore del pennello con tutti i colori già presenti sul canvas. Gli stili predefiniti comprendono il colore uniforme, nessun colore e vari modelli di linea e di tratteggio.

Per cambiare lo stile di un pennello, occorre impostare la sua proprietà *Style* ad uno dei valori predefiniti: *bsSolid*, *bsClear*, *bsHorizontal*, *bsVertical*, *bsFDiagonal*, *bsBDiagonal*, *bsCross* o *bsDiagCross*.

Questo esempio imposta gli stili del pennello condividendo un gestore per gli eventi *OnClick* relativi ad un gruppo di otto pulsanti per gli stili di pennello. Tutti e otto i pulsanti vengono selezionati, viene impostato Object Inspector|Events|*OnClick*, e il gestore *OnClick* viene chiamato *SetBrushStyle*. Il codice del gestore è:

```
void __fastcall TForm1::SetBrushStyle(TObject *Sender)
{
  if (Sender == SolidBrush)
    Canvas->Brush->Style = bsSolid;
  else if (Sender == ClearBrush)
    Canvas->Brush->Style = bsClear;
  else if (Sender == HorizontalBrush)
    Canvas->Brush->Style = bsHorizontal;
  else if (Sender == VerticalBrush)
    Canvas->Brush->Style = bsVertical;
  else if (Sender == FDiagonalBrush)
    Canvas->Brush->Style = bsFDiagonal;
  else if (Sender == BDiagonalBrush)
    Canvas->Brush->Style = bsBDiagonal;
  else if (Sender == CrossBrush)
    Canvas->Brush->Style = bsCross;
  else if (Sender == DiagCrossBrush)
    Canvas->Brush->Style = bsDiagCross;
}
```

Il gestore di evento visto in precedenza potebbe essere ulteriormente ridotto ponendo le costanti relative allo stile del pennello nelle proprietà *Tag* dei pulsanti

relativi allo stile del pennello. In questo caso, il codice dell'evento sarebbe simile al seguente:

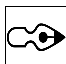
```
void __fastcall TForm1::SetBrushStyle(TObject *Sender)
{
   if (Sender->InheritsFrom (__classid(TSpeedButton))
    Canvas->Brush->Style = (TBrushStyle) ((TSpeedButton *)Sender)->Tag;
}
```

Impostazione della proprietà Bitmap del pennello

La proprietà *Bitmap* di un pennello permette di specificare un'immagine bitmap da usare come modello per riempire le figure e le altre aree.

L'esempio seguente carica una bitmap da un file e la assegna al pennello del canvas di Form1:

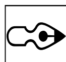
```
BrushBmp->LoadFromFile("MyBitmap.bmp");
  Form1->Canvas->Brush->Bitmap = BrushBmp;
  Form1->Canvas->FillRect(Rect(0,0,100,100));
```



Il pennello non si impossessa dell'oggetto *Bitmap* assegnato alla sua proprietà *Bitmap*. Bisogna assicurarsi che l'oggetto Bitmap rimanga valido per tutta la durata del pennello, e che successivamente venga liberato.

Lettura e impostazione dei pixel

Si noterà che ogni canvas ha una proprietà indicizzata *Pixels* che rappresenta i singoli punti colorati che costituiscono l'immagine sul canvas. Raramente è necessario accedere direttamente a *Pixels*, essa è disponibile solo per eseguire facilmente piccole azioni, come le ricerca o l'impostazione del colore di un pixel.



L'impostazione e la rilevazione dei singoli pixel sono migliaia di volte più lente dell'esecuzione di operazioni grafiche sulle regioni. Non usare la proprietà Pixel dell'array per accedere ai pixel dell'immagine di un generico array. Per l'accesso ad elevate prestazioni ai pixel dell'immagine, consultare la proprietà *TBitmap*:*ScanLine*.

Uso dei metodi Canvas per disegnare oggetti grafici

Questa sezione mostra come usare alcuni metodi comuni per disegnare oggetti grafici. Gli argomenti trattati sono:

- Disegno di linee e spezzate.
- Disegno di figure.
- Disegno di rettangoli arrotondati.
- Disegno di poligoni.

Disegno di linee e spezzate

Un canvas può disegnare linee rette e spezzate. Una linea retta è solo una riga di pixel compresi tra due punti. Una spezzata è una serie di linee rette, collegate alle estremità. Il canvas disegna tutte le linee usando la sua penna.

Disegno di linee

Per disegnare una linea retta su un canvas, si deve usare il metodo *LineTo* del canvas.

LineTo disegna una linea dalla posizione attuale della penna al punto specificato e rende il punto finale della linea la posizione attiva. Il canvas disegna la linea usando la sua penna.

Per esempio, il metodo seguente disegna diagonalmente linee incrociate in una scheda ogni volta che la scheda viene dipinta:

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{
  Canvas->MoveTo(0,0);
  Canvas->LineTo(ClientWidth, ClientHeight);
  Canvas->MoveTo(0, ClientHeight);
  Canvas->LineTo(ClientWidth, 0);
}
```

Disegno di spezzate

Oltre alle singole linee, il canvas può disegnare anche spezzate, cioè gruppi di un numero qualunque di segmenti consecutivi.

Per disegnare una spezzata su un canvas, si deve chiamare il metodo *Polyline* del canvas.

Il parametro passato al metodo *Polyline* è un array di punti.Si può pensare ad una a spezzata come all'esecuzione di un *MoveTo* sul primo punto e di un *LineTo* su ciascun punto successivo. Per disegnare più linee, *Polyline* è più rapido dell'uso dei metodi *MoveTo* e *LineTo*, in quanto viene eliminato l'overhead derivante dalle successive chiamate.

Il metodo seguente, per esempio, disegna un rombo in una scheda:

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{
  TPoint vertices[5];
  vertices[0] = Point(0, 0);
  vertices[1] = Point(50, 0);
  vertices[2] = Point(75, 50);
  vertices[3] = Point(25, 50);
  vertices[4] = Point(0, 0);
  Canvas->Polyline(vertices, 4);
}
```

Si noti che l'ultimo parametro di *Polyline* è l'indice dell'ultimo punto, e non il numero di punti.

Disegno di figure

I canvas hanno metodi per disegnare tipi differenti di figure. Il canvas disegna il contorno di una figura con la sua penna, quindi ne riempie l'interno con il suo pennello. La linea che rappresenta il bordo della figura è controllata dall'oggetto *Pen* attivo.

In questa sezione vengono trattati i seguenti argomenti:

- Disegno di rettangoli ed ellissi.
- Disegno di rettangoli arrotondati.
- Disegno di poligoni.

Disegno di rettangoli ed ellissi

Per disegnare un rettangolo o un'ellisse su un canvas, si deve chiamare il metodo *Rectangle* o il metodo *Ellipse* del canvas, passando le coordinate di un rettangolo di contorno.

Il metodo *Rectangle* disegna il rettangolo di contorno; *Ellipse* disegna un'ellisse che tocca tutti i lati del rettangolo.

Il metodo seguente disegna un rettangolo riempiendo il quadrante superiore sinistro di una scheda, quindi disegna un'ellisse nella stessa area:

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{
  Canvas->Rectangle(0, 0, ClientWidth/2, ClientHeight/2);
  Canvas->Ellipse(0, 0, ClientWidth/2, ClientHeight/2);
}
```

Disegno di rettangoli arrotondati

Per disegnare un rettangolo arrotondato su un canvas, si deve chiamare il metodo *RoundRect* del canvas.

I primi quattro parametri passati a *RoundRect* sono un rettangolo di contorno, proprio come avviene per il metodo *Rectangle* o per il metodo *Ellipse*. *RoundRect* accetta altri due parametri che indicano come disegnare gli angoli arrotondati.

Il metodo seguente, per esempio, disegna un rettangolo arrotondato nel quadrante superiore sinistro di una scheda, arrotondando gli angoli come sezioni di un cerchio con un diametro di 10 pixel:

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{
  Canvas->RoundRect(0, 0, ClientWidth/2, ClientHeight/2, 10, 10);
}
```

Disegno di poligoni

Per disegnare un poligono con un numero qualsiasi di lati su un canvas, si deve chiamare il suo metodo *Polygon*.

Polygon accetta come unico parametro un array di punti. Questi vengono collegati con la penna, e l'ultimo punto viene collegato al primo per chiudere il poligono. Dopo aver disegnato le linee, *Polygon* usa il pennello per riempire l'area all'interno del poligono.

Ad esempio, il codice seguente disegna un triangolo rettangolo nella metà inferiore sinistra di una scheda:

```
void __fastcall TForm1::FormPaint(TObject *Sender)
{
  TPoint vertices[3];
```

```
  vertices[0] = Point(0, 0);
  vertices[1] = Point(0, ClientHeight);
  vertices[2] = Point(ClientWidth,ClientHeight);
  Canvas->Polygon(vertices,2);
}
```

Gestione di più oggetti di disegno in un'applicazione

Normalmente, i vari metodi di disegno (rettangolo, figura, linea, e così via) sono disponibili nella barra strumenti e nel riquadro dei pulsanti. Le applicazioni possono rispondere ai clic sui pulsanti di scelta rapida per impostare gli oggetti di disegno desiderati. Questa sezione illustra i seguenti argomenti:

- Come tenere la traccia dello strumento di disegno da usare.
- Modifica dello strumento con i pulsanti di scelta rapida.
- Uso degli strumenti di disegno.

Come tenere la traccia dello strumento di disegno da usare

Un programma grafico ha la necessità di tenere la traccia del tipo di strumento di disegno (come una linea, un rettangolo, un'ellisse o un rettangolo arrotondato) che un utente può aver bisogno di usare in un istante ben preciso. Di solito, si dovrebbe usare il tipo enumerativo di C++ per elencare gli strumenti disponibili. Poiché un tipo enumerativo è anche una dichiarazione di tipo, è possibile sfruttare il controllo sui tipi di C++ per essere certi di assegnare *solo* quei valori specifici.

Per esempio, il codice seguente dichiara un tipo enumerativo per ogni strumento di disegno disponibile in un'applicazione grafica:

```
typedef enum {dtLine, dtRectangle, dtEllipse, dtRoundRect} TDrawingTool;
```

A una variabile di tipo *TDrawingTool* può essere assegnata solo una delle costanti *dtLine*, *dtRectangle*, dtEllipse o *dtRoundRect*.

Per convenzione, gli identificatori di tipo iniziano con la lettera *T*, mentre i gruppi di costanti simili (come quelle che costituiscono un tipo enumerativo) iniziano con un prefisso di due lettere (come *dt* per "drawing tool").

Nel codice seguente, un campo aggiunto ad una scheda tiene traccia dello strumento di disegno usato:

```
enum TDrawingTool {dtLine, dtRectangle, dtEllipse, dtRoundRect};

class TForm1 : public TForm
{
__published:  // IDE-managed Components
  void __fastcall FormMouseDown(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y);
  void __fastcall FormMouseMove(TObject *Sender, TShiftState Shift, int X,
  int Y);
  void __fastcall FormMouseUp(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y);
private:// User declarations
```

```
public:// User declarations
  __fastcall TForm1(TComponent* Owner);
  bool Drawing;  //field to track whether button was pressed
  TPoint Origin, MovePt;  // fields to store points
  TDrawingTool DrawingTool;  // field to hold current tool
};
```

Modifica degli strumenti con i pulsanti di scelta rapida

Ogni strumento di disegno ha bisogno di un gestore di evento *OnClick* associato. Si supponga che l'applicazione abbia un pulsante della barra strumenti per ciascuno dei quattro strumenti di disegno: linea, rettangolo, ellisse e rettangolo arrotondato. Occorrerà collegare i seguenti gestori di evento agli eventi *OnClick* dei quattro pulsanti degli strumenti di disegno, impostando *DrawingTool* al valore appropriato per ciascuno di loro:

```
void __fastcall TForm1::LineButtonClick(TObject *Sender)  // LineButton
{
  DrawingTool = dtLine;
}
void __fastcall TForm1::RectangleButtonClick(TObject *Sender) // RectangleButton
{
  DrawingTool = dtRectangle;
}
void __fastcall TForm1::EllipseButtonClick(TObject *Sender)  // EllipseButton
{
  DrawingTool = dtEllipse;
}
void __fastcall TForm1::RoundedRectButtonClick(TObject *Sender) // RoundRectBtn
{
  DrawingTool = dtRoundRect;
}
```

Uso degli strumenti di disegno

Ora che si è in grado di stabilire quale strumento usare, bisogna indicare come disegnare le diverse figure. Gli unici metodi che eseguono qualsiasi disegno sono i gestori mouse-move e mouse-up, mentre il codice disegna solo linee, indipendentemente dallo strumento selezionato.

Per usare i strumenti di disegno diversi, il codice ha bisogno che venga specificato come disegnare, sulla base allo strumento selezionato. Si deve aggiungere questa istruzione al gestore di evento di ciascuno strumento.

In questa sezione vengono trattati i seguenti argomenti:

- Disegno di figure.
- Condivisione del codice tra i gestori di evento.

Disegno di figure

Disegnare figure è facile quanto disegnare linee. Per tracciarne una è sufficiente una singola istruzione; sono necessarie solo le coordinate.

Il codice seguente è una riscrittura del gestore di evento *OnMouseUp* che disegna figure per tutti e quattro gli strumenti:

```
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button,
                                    TShiftState Shift, int X, int Y){
  switch (DrawingTool)
  {
    case dtLine:
      Canvas->MoveTo(Origin.x, Origin.y);
      Canvas->LineTo(X, Y);
      break;
    case dtRectangle:
      Canvas->Rectangle(Origin.x, Origin.y, X, Y);
      break;
    case dtEllipse:
      Canvas->Ellipse(Origin.x, Origin.y, X, Y);
      break;
    case dtRoundRect:
      Canvas->Rectangle(Origin.x, Origin.y, X, Y, (Origin.x - X)/2,
                  (Origin.y - Y)/2);
      break;
  }
  Drawing = false;
}
```

Naturalmente, occorre anche aggiornare il gestore *OnMouseMove* per disegnare figure:

```
void __fastcall TForm1::FormMouseMove(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
  if (Drawing)
  {
    Canvas->Pen->Mode = pmNotXor;        // use XOR mode to draw/erase
    switch (DrawingTool)
    {
      case dtLine:
        Canvas->MoveTo(Origin.x, Origin.y);
        Canvas->LineTo(MovePt.x, MovePt.y);
        Canvas->MoveTo(Origin.x, Origin.y);
        Canvas->LineTo(X, Y);
        break;
      case dtRectangle:
        Canvas->Rectangle(Origin.x, Origin.y, MovePt.x, MovePt.y);
        Canvas->Rectangle(Origin.x, Origin.y, X, Y);
        break;
      case dtEllipse:
        Canvas->Ellipse(Origin.x, Origin.y, MovePt.x, MovePt.y);
        Canvas->Ellipse(Origin.x, Origin.y, X, Y);
        break;
      case dtRoundRect:
        Canvas->Rectangle(Origin.x, Origin.y, MovePt.x, MovePt.y,
                  (Origin.x - MovePt.x)/2,(Origin.y - MovePt.y)/2);
        Canvas->Rectangle(Origin.x, Origin.y, X, Y,
                  (Origin.x - X)/2, (Origin.y - Y)/2);
```

```
        break;
    }
    MovePt = Point(X, Y);
  }
  Canvas->Pen->Mode = pmCopy;
}
```

Normalmente, tutto il codice ripetitivo dell'esempio sopra riportato dovrebbe essere inserito in una routine separata.La sezione successiva mostra tutto il codice per il disegno delle figure in una singola routine che i gestori di evento del *mouse* possono chiamare.

Condivisione del codice tra gestori di evento

Quando ci si rende conto del fatto che molti gestori di evento usano lo stesso codice, si può rendere l'applicazione più efficiente spostando il codice ripetuto in una routine che tutti i gestori di evento possono condividere.

Per aggiungere un metodo ad una scheda:

1 Aggiungere la dichiarazione del metodo all'oggetto scheda.

   La dichiarazione può essere aggiunta nelle parti **public** o **private** alla fine della dichiarazione dell'oggetto scheda. Se il codice condivide solo i dettagli di gestione di alcuni eventi, probabilmente è più sicuro rendere il metodo condiviso **private**.

2 Scrivere l'implementazione del metodo nel file .cpp della unit della scheda.

L'header per l'implementazione del metodo deve corrispondere esattamente alla dichiarazione, con gli stessi parametri nello stesso ordine.

Il codice seguente aggiunge alla scheda un metodo *DrawShape*, che viene chiamato da ciascuno dei gestori. Innanzi tutto, la dichiarazione di *DrawShape* viene aggiunta alla dichiarazione dell'oggetto scheda:

```
enum TDrawingTool {dtLine, dtRectangle, dtEllipse, dtRoundRect};

class TForm1 : public TForm
{
__published:  // IDE-managed Components
  void __fastcall FormMouseDown(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y);
  void __fastcall FormMouseMove(TObject *Sender, TShiftState Shift, int X,
  int Y);
  void __fastcall FormMouseUp(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y);
private:// User declarations
  void __fastcall DrawShape(TPoint TopLeft, TPoint BottomRight, TPenMode AMode);
public:// User declarations
  __fastcall TForm1(TComponent* Owner);
  bool Drawing;  //field to track whether button was pressed
  TPoint Origin, MovePt;  // fields to store points
  TDrawingTool DrawingTool;  // field to hold current tool

};
```

Quindi, l’implementazione di *DrawShape* viene scritta nel file .cpp della unit della scheda:

```
void __fastcall TForm1::DrawShape(TPoint TopLeft, TPoint BottomRight,
  TPenMode AMode)
{
  Canvas->Pen->Mode = AMode;
  switch (DrawingTool)
  {
    case dtLine:
      Canvas->MoveTo(TopLeft.x, TopLeft.y);
      Canvas->LineTo(BottomRight.x, BottomRight.y);
      break;
    case dtRectangle:
      Canvas->Rectangle(TopLeft.x, TopLeft.y, BottomRight.x, BottomRight.y);
      break;
    case dtEllipse:
      Canvas->Ellipse(TopLeft.x, TopLeft.y, BottomRight.x, BottomRight.y);
      break;
    case dtRoundRect:
      Canvas->Rectangle(TopLeft.x, TopLeft.y, BottomRight.x, BottomRight.y,
                (TopLeft.x - BottomRight.x)/2,(TopLeft.y - BottomRight.y)/2);
      break;
  }
}
```

Gli altri gestori di evento vengono modificati per chiamare *DrawShape*.

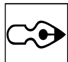
```
void __fastcall TForm1::FormMouseUp(TObject *Sender)
{
  DrawShape(Origin, Point(X,Y), pmCopy); // draw the final shape
  Drawing = false;
}

void __fastcall TForm1::FormMouseMove(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
  if (Drawing)
  {
    DrawShape(Origin, MovePt, pmNotXor); // erase previous shape
    MovePt = Point(X, Y);
    DrawShape(Origin, MovePt, pmNotXor); // draw current shape
  }
}
```

Disegno su un grafico

Non sono necessari componenti per manipolare gli oggetti grafici dell’applicazione. È possibile costruire, prelevare, salvare e distruggere oggetti grafici senza dover necessariamente disegnare qualcosa sullo schermo. Infatti, raramente le applicazioni disegnano direttamente su una scheda. Più spesso un’applicazione opera sui grafici e poi utilizza un componente controllo immagine per visualizzare il grafico su una scheda.

Una volta spostato il disegno dell'applicazione nel grafico del controllo immagine, è facile aggiungere le funzioni di stampa, di clipboard, di caricamento e salvataggio per tutti gli oggetti grafici. Gli oggetti grafici possono essere file bitmap, metafile, icone o qualsiasi altra classe di grafici installati, come, ad esempio, i grafici jpeg.



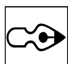
Poiché si disegna su un'immagine fuori schermo come un canvas *TBitmap*, l'immagine non viene visualizzata finché un controllo non copia da una bitmap sul canvas del controllo.Cioè, quando si disegnano bitmap e si assegnano ad un controllo immagine, l'immagine appare solo quando il controllo ha l'opportunità per elaborare il messaggio Paint. Se, invece, si disegna direttamente sulla proprietà canvas di un controllo, l'oggetto *Picture* viene visualizzato immediatamente.

Come far scorrere i grafici

Il grafico non deve avere le stesse dimensioni della scheda: può essere più piccolo o più grande. Aggiungendo alla scheda un controllo casella di scorrimento e collocando l'immagine grafica al suo interno, è possibile visualizzare grafici che sono più grandi della scheda o addirittura più grandi dello schermo. Per aggiungere un grafico con scorrimento, occorre prima aggiungere un componente *TScrollBox* e quindi aggiungere il controllo immagine.

Aggiunta di un controllo immagine

Un controllo immagine è un componente contenitore che consente di visualizzare oggetti *Bitmap*. Un controllo immagine serve per contenere una bitmap che non necessariamente viene visualizzata sempre, o che un'applicazione ha bisogno di usare per generare altre figure.



Il paragrafo"Aggiunta di grafici ai controlli" a pagina 6-12 illustra come usare i grafici nei controlli.

Collocazione del controllo

Un controllo immagine può essere collocato in qualsiasi punto di una scheda. Se si desidera sfruttare la capacità del controllo immagine di autodimensionarsi in funzione della figura, occorre impostare solo l'angolo superiore sinistro. Se il controllo immagine è un contenitore non visibile per una bitmap, si può collocarlo in un punto qualsiasi, proprio come se fosse un componente non visuale.

Se si rilascia il controllo immagine su una casella di scorrimento già allineata all'area client della scheda, ciò garantisce che la casella aggiungerà tutte le barre di scorrimento necessarie per accedere alle parti fuori schermo dell'immagine. Dopo di che si deve impostare la proprietà del controllo immagine.

Impostazione delle dimensioni iniziali della bitmap

Quando si colloca un controllo immagine, questo è solo un contenitore. Tuttavia, in fase di progettazione, è possibile impostare la proprietà *Picture* del controllo immagine in modo da contenere un grafico statico. Il controllo può caricare la sua immagine da un file anche in fase di esecuzione, come descritto nel paragrafo "Caricamento e salvataggio dei file grafici" a pagina 10-20.

Per creare una bitmap vuota quando viene avviata l'applicazione:

1 Collegare un gestore all'evento *OnCreate* per la scheda che contiene l'immagine.

2 Creare un oggetto Bitmap, e assegnarlo alla proprietà *Picture->Graphic* del controllo immagine.

In questo esempio l'immagine si trova nella scheda principale dell'applicazione, *Form1*, cosicché il codice collega un gestore all'evento *OnCreate* di *Form1*:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Graphics::TBitmap *Bitmap = new Graphics::TBitmap();   // create the bitmap object
  Bitmap->Width = 200;               // assign the initial width...
  Bitmap->Height = 200;              // ...and the initial height
  Image->Picture->Graphic = Bitmap;  // assign the bitmap to the image control
  delete Bitmap;                     // free the bitmap object
}
```

L'assegnazione della bitmap alla proprietà *Graphic* dell'oggetto picture copia la bitmap nell'oggetto Picture. Tuttavia, l'oggetto immagine non assume la proprietà della bitmap, e pertanto, dopo avere fatto l'assegnazione, sarà necessario liberarlo.

Se a questo punto si esegue l'applicazione, l'area client della scheda avrà una regione bianca, che rappresenta la bitmap. Se le finestra viene dimensionata in modo che l'area client non possa visualizzare l'intera immagine, la casella di scorrimento mostrerà automaticamente le barre di scorrimento per consentire la visualizzazione del resto dell'immagine. Ma, se si cerca di disegnare sull'immagine, non si ottiene alcun grafico, perché l'applicazione sta ancora disegnando sulla scheda, che ora si trova dietro l'immagine e la casella di scorrimento.

Disegno sulla bitmap

Per disegnare su una bitmap, si deve usare il canvas del controllo immagine e collegare i gestori di evento del *mouse* a quelli corrispondenti del controllo immagine. Di solito, si utilizzeranno operazioni di regione (riempienti, rettangoli, spezzate e così via). Questi sono metodi di disegno rapidi ed efficaci.

Un modo efficiente per disegnare immagini quando si ha bisogno di accedere ai singoli pixel, consiste nel ricorrere alla proprietà *ScanLine* della bitmap. Per un uso generico, è possibile impostare il formato dei pixel della bitmap a 24 bit e quindi trattare il puntatore restituito da *ScanLine* come un array di RGB. Altrimenti, sarà necessario conoscere il formato nativo della proprietà *ScanLine*. Questo esempio mostra come usare *ScanLine* per ricevere i pixel una riga alla volta.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Graphics::TBitmap *pBitmap = new Graphics::TBitmap();
// This example shows drawing directly to the Bitmap
  Byte *ptr;
  try
  {
    pBitmap->LoadFromFile("C:\\Program Files\\Borland\\CBuilder\\Images\\Splash\\256color\\
factory.bmp ");
    for (int y = 0; y < pBitmap->Height; y++)
    {
```

```
        ptr = pBitmap->ScanLine[y];
        for (int x = 0; x < pBitmap->Width; x++)
          ptr[x] = (Byte)y;
      }
      Canvas->Draw(0,0,pBitmap);
    }
    catch (...)
    {
      ShowMessage("Could not load or alter bitmap");
    }
    delete pBitmap;
  }
```

Nel caso di applicazioni multipiattaforma, modificare il codice specifico per Windows e per la VCL in modo che l'applicazione possa esere eseguita in ambiente Linux. Ad esempio, nei nomi dei percorsi in Linux si utilizza come delimitatore una barra diritta. Per ulteriori informazioni sulle applicazioni CLX e multipiattaforma , vedere il Capitolo 14, "Sviluppo di applicazioni multipiattaforma".

Caricamento e salvataggio dei file grafici

Le immagini grafiche che esistono solo per la durata dell'esecuzione di un'applicazione hanno un valore molto limitato. Spesso, si preferisce usare la stessa figura ogni volta o salvare una figura creata per usarla successivamente. Il componente image facilita il caricamento delle immagini da un file e il loro salvataggio.

I componenti usati per caricare, salvare e sostituire immagini grafiche supportano molti formati grafici, compresi i file bitmap, i metafiles, i glifi, e così via. Essi supportano anche le classi grafiche installabili.

Il modo di caricare e di salvare i file grafici è simile a quello di qualsiasi altro file e viene descritto nelle seguenti sezioni:

- Caricamento di un'immagine da un file.
- Salvataggio di un'immagine in un file.
- Sostituzione dell'immagine.

Caricamento di un'immagine da un file

L'applicazione deve fornire la possibilità di caricare un'immagine da un file se ha bisogno di modificarla o se si desidera memorizzarla in un file esterno, in modo che un utente o un'altra applicazione possa modificarla.

Per caricare un file grafico in un controllo immagine, si deve chiamare il metodo *LoadFromFile* dell'oggetto *Picture* del controllo immagine.

Il codice seguente riceve un nome di file da una finestra di dialogo per l'apertura dei file, e quindi carica quel file in un controllo immagine di nome *Image*:

```
void __fastcall TForm1::Open1Click(TObject *Sender)
{
  if (OpenPictureDialog1->Execute())
```

```
  {
    CurrentFile = OpenPictureDialog1->FileName;
    Image->Picture->LoadFromFile(CurrentFile);
  }
}
```

Salvataggio di un'immagine in un file

L'oggetto Picture della libreria VCL può caricare e salvare grafici in diversi formati, ed è possibile creare e registrare propri formati di file grafici, in modo che gli oggetti picture possano caricarli e memorizzarli.

Per salvare il contenuto di un controllo immagine in un file, si deve chiamare il metodo *SaveToFile* dell'oggetto *Picture* del controllo immagine.

Il metodo *SaveToFile* richiede il nome del file da salvare. Se l'immagine è stata appena creata, è probabile che non abbia un nome di file; oppure, può darsi che un utente voglia salvare una figura esistente in un altro file. In ogni caso l'applicazione ha bisogno di ricevere un nome di file dall'utente prima del salvataggio, come viene mostrato nella successiva sezione.

La seguente coppia di gestori di evento, collegata rispettivamente alle voci di menu File|Save e File|Save As, gestisce il nuovo salvataggio dei file ai quali è già stato assegnato un nome, dei file ancora senza nome e dei file esistenti con nomi nuovi.

```
void __fastcall TForm1::Save1Click(TObject *Sender)
{
  if (!CurrentFile.IsEmpty())
    Image->Picture->SaveToFile(CurrentFile);   // save if already named
else SaveAs1Click(Sender);                     // otherwise get a name
}

void __fastcall TForm1::SaveAs1Click(TObject *Sender)
{
  if (SaveDialog1->Execute())              // get a file name
  {
    CurrentFile = SaveDialog1->FileName;  // save user-specified name
    Save1Click(Sender);                   // then save normally
  }
}
```

Sostituzione dell'immagine

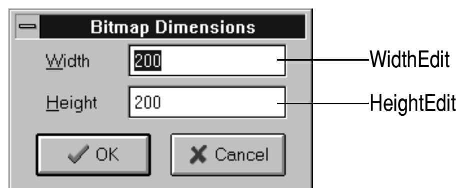
In qualsiasi momento è possibile sostituire la e figura di un controllo immagine. Se si assegna un nuovo grafico ad un'immagine che ne ha già uno, il nuovo grafico sostituisce quello esistente.

Per sostituire la figura di un controllo immagine, si deve assegnare un nuovo grafico all'oggetto *Picture* del controllo immagine.

Il processo adottato per la creazione del nuovo grafico è identico a quello utilizzato per creare il grafico iniziale (consultare "Impostazione delle dimensioni iniziali della bitmap" a pagina 10-18), ma in questo caso si deve fornire anche un modo che consenta alperché l'utente di scegliere una dimensione diversa da quella standard

usata per il grafico iniziale. Un semplice modo per fornire questa opzione consiste nel presentare una finestra di dialogo, come quella della Figura 10.1.

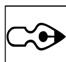
Figura 10.1 Finestra di dialogo Bitmap Dimension dalla unit BMPDlg



Questa particolare finestra di dialogo viene creata nella unit *BMPDlg* che fa parte del progetto *GraphEx* (nella directory EXAMPLES\DOC\GRAPHEX).

Avendo questa finestra di dialogo nel progetto, aggiungere un'istruzione include relativa a *BMPDlg*.hpp nel file .cpp della scheda principale. Quindi, si può collegare un gestore all'evento *onClick* della voce di menu File | New. Ecco un esempio:

```
void __fastcall TForm1::New1Click(TObject *Sender)
{
     Graphics::TBitmap *Bitmap;
  // make sure focus is on width field
     NewBMPForm->ActiveControl = NewBMPForm->WidthEdit;
  // initialize to current dimensions as default ...
     NewBMPForm->WidthEdit->Text = IntToStr(Image->Picture->Graphic->Width);
     NewBMPForm->HeightEdit->Text = IntToStr(Image->Picture->Graphic->Height);
     if (NewBMPForm->ShowModal() != IDCANCEL){        // if user does not cancel dialog...
        Bitmap = new Graphics::TBitmap();             // create a new bitmap object
        // use specified dimensions
        Bitmap->Width = StrToInt(NewBMPForm->WidthEdit->Text);
        Bitmap->Height = StrToInt(NewBMPForm->HeightEdit->Text);
        Image->Picture->Graphic = Bitmap;            // replace graphic with new bitmap
        CurrentFile = EmptyStr;                      //indicate unnamed file
        delete Bitmap;
     }
}
```



L'assegnazione di una nuova bitmap alla proprietà *Graphic* dell'oggetto Picture provoca la copia del nuovo grafico nell'oggetto ma l'oggetto non ne prende possesso. L'oggetto Picture gestisce il proprio oggetto grafico interno. . A causa di ciò, il codice precedente libera l'oggetto bitmap dopo avere fatto l'assegnazione.

Uso della clipboard con i grafici

È possibile usare la clipboard di Windows per copiare e incollare grafici all'interno delle applicazioni o per scambiare grafici con altre applicazioni. L'oggetto Clipboard della VCL semplifica la gestione di diversi tipi di informazione, compresi i grafici.

Prima di poter usare l'oggetto clipboard nell'applicazione, è necessario aggiungere ad ogni file .cpp che dovrà accedere ai dati della clipboard un'istruzione include per il file Clipbrd.hpp.

Per le applicazioni multipiattaforma, i dati memorizzati nella clipboard utilizzando la CLX vengono memorizzati come un tipo MIME a cui è associato un oggetto *TStream*. Nella CLX sono previste le seguenti costanti predefinite per i tipi MIME seguenti.

Tabella 10.4 Tipi MIME e costanti della CLX

Tipo MIME	Costante CLX
'image/delphi.bitmap'	SDelphiBitmap
'image/delphi.component'	SDelphiComponent
'image/delphi.picture'	SDelphiPicture
'image/delphi.drawing'	SDelphiDrawing

Operazioni di copia di grafici nella clipboard

Nella clipboard è possibile copiare qualsiasi immagine, compreso il contenuto dei controlli image. Una volta nella clipboard, l'immagine è disponibile per tutte le applicazioni.

Per copiare un'immagine nella clipboard, bisogna assegnarla all'oggetto clipboard usando il metodo *Assign*.

Questo codice mostra come copiare un'immagine da un controllo immagine di nome *Image* nella clipboard in risposta ad un clic sulla voce di menu Edit|Copy:

```
void __fastcall TForm1::Copy1Click(TObject *Sender)
{
  Clipboard()->Assign(Image->Picture);
}
```

Operazioni di taglio di grafici nella clipboard

Le operazioni di taglio di un grafico negli Appunti sono analoghe alle operazioni di copia, con la differenza che il grafico viene eliminato dalla sorgente.

Per tagliare un grafico da una figura nella clipboard, occorre prima copiarlo nella clipboard, quindi si deve eliminare l'originale.

Nella maggioranza dei casi l'unico problema nel taglio è come mostrare che l'immagine originale è stata eliminata. L'impostazione dell'area sul colore bianco è una soluzione diffusa, come viene mostrato nel codice seguente che collega un gestore all'evento *OnClick* della voce di menu Edit|Cut:

```
void __fastcall TForm1::Cut1Click(TObject *Sender)
{
  TRect ARect;
  Copy1Click(Sender);          // copy picture to clipboard
  Image->Canvas->CopyMode = cmWhiteness; // copy everything as white
  ARect = Rect(0, 0, Image->Width, Image->Height); // get dimensions of image
  Image->Canvas->CopyRect(ARect, Image->Canvas, ARect); // copy bitmap over self
  Image->Canvas->CopyMode = cmSrcCopy; // restore default mode
}
```

Operazioni di incollaggio di grafici dalla clipboard

Se la clipboard contiene un grafico bitmap, è possibile incollarlo in qualsiasi oggetto immagine, compresi i controlli immagine e la superficie di una scheda.

Per incollare un grafico dalla clipboard:

1 Chiamare il metodo *HasFormat* della clipboard (se si usa la VCL) oppure il metodo *Provides* (se si usa la CLX) per controllare se contiene un'immagine grafica.

   HasFormat (o *Provides* in CLX) è una funzione Boolean. Restituisce **true** se la clipboard contiene un elemento del tipo specificato nel parametro. Per verificare se c'è un grafico in ambiente Windows, basta passare *CF_BITMAP*. In applicazioni multipiattaforma, si passa *SDelphiBitmap*.

2 Assegnare la clipboard alla destinazione.

Questo codice mostra come incollare una figura dalla clipboard in un controllo immagine in risposta ad un clic su una voce di menu Edit|Paste:

```
void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  Graphics::TBitmap *Bitmap;
  if (Clipboard()->HasFormat(CF_BITMAP)){
   Image1->Picture->Bitmap->Assign(Clipboard());
  }
}
```

Lo stesso esempio in un programma CLX multipiattaforma sarebbe simile al seguente:

```
void __fastcall TForm1::Paste1Click(TObject *Sender)
{
  QGraphics::TBitmap *Bitmap;
  if (Clipboard()->Provides(SDelphiBitmap)){
   Image1->Picture->Bitmap->Assign(Clipboard());
  }
}
```

Il grafico nela clipboard può provenire da questa applicazione, oppure potrebbe anche essere stato copiato da un'altra applicazione, come Microsoft Paint. In questo caso non è necessario controllare il formato della clipboard, in quanto il menu Paste viene disattivato quando la clipboard non contiene un formato supportato.

Esempio di effetto elastico

Questo esempio descrive i dettagli implementativi dell'effetto "elastico" in un'applicazione grafica che segue la traccia dei movimenti del mouse, man mano che l'utente disegna un grafico in fase di esecuzione. Il codice di esempio riportato in questa sezione è tratto da un'applicazione di esempio collocata nella directory Examples\Doc\GraphEx. L'applicazione disegna linee e figure nel canvas di una finestra in risposta ai clic e alle operazioni di trascinamento: la pressione di un pulsante del mouse button avvia il disegno, mentre il suo rilascio ne determina la fine.

All'inizio, il codice di esempio mostra come disegnare sulla superficie della scheda principale. Gli esempi successivi mostrano come disegnare su una bitmap.

I seguenti argomenti descrivono l'esempio:

- Risposta al mouse.
- Aggiunta di un campo ad un oggetto scheda per seguire le tracce delle azioni del mouse.
- Affinamento del disegno delle linee.

Risposta al mouse

L'applicazione può rispondere alle azioni del mouse: pulsante-mouse giù, spostamento mouse e pulsante-mouse su. Essa può rispondere anche ad un clic (un'operazione completa di pressione e rilascio nello stesso punto), che può essere generato da una combinazione di tasti (come le pressione di *Invio* in una finestra di dialogo modale).

In questa sezione vengono trattati i seguenti argomenti:

- Cosa c'è in un evento mouse.
- Risposta ad un'azione pulsante mouse giù.
- Risposta ad un'azione mouse-su.
- Risposta ad uno spostamento del mouse.

Cosa c'è in un evento mouse??

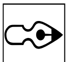
C++Builder ha tre eventi *mouse*: *OnMouseDown*,*OnMouseMove* e *OnMouseUp*.

Quando un'applicazione rileva un'azione del mouse, chiama il gestore di evento che è stato definito per l'evento corrispondente, passando cinque parametri. Le informazioni di questi parametri devono essere utilizzate per personalizzare le risposte agli eventi. I cinque parametri sono i seguenti:

Tabella 10.5 Parametri degli eventi mouse

Parametro	Significato
Sender	L'oggetto che ha rilevato l'azione del mouse
Button	Indica quale pulsante del mouse è stato coinvolto: *mbLeft*, *mbMiddle*, or *mbRight*
Shift	Indica lo stato dei tasti *Alt*, *Ctrl*, e *Maisc* al momento dell'azione del mouse
X, *Y*	Le coordinate del punto in cui si è verificato l'evento

Nella maggioranza dei casi le coordinate devono essere passate ad un gestore di evento del mouse, ma talvolta è anche necessario controllare *Button* per determinare quale pulsante del mouse ha causato l'evento.



C++Builder usa gli stessi criteri di Microsoft Windows nel determinare quale pulsante del *mouse* è stato premuto. Pertanto, se sono stati invertiti i pulsanti del *mouse* predefiniti "principale" e "secondario" (in modo che il pulsante destro sia quello principale), un clic sul pulsante principale (destro) registrerà *mbLeft* come valore del parametro *Button*.

Risposta ad un'azione mouse-down

Ogni volta che l'utente preme un pulsante sul mouse, l'oggetto sul quale si trova il puntatore riceve un evento *OnMouseDown*. L'oggetto quindi può rispondere all'evento.

Per rispondere ad un'azione mouse-down, si deve collegare un gestore all'evento *OnMouseDown*.

C++Builder genera un gestore vuoto per un evento mouse-down sulla scheda:

```
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{

}
```

Risposta ad un'azione mouse-down

Il codice seguente visualizza sulla scheda la stringa 'Here!' alla posizione in cui è stato fatto clic con il mouse:

```
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
  Canvas->TextOut(X, Y, "Here!");// write text at (X, Y)
}
```

Durate l'esecuzione dell'applicazione, se si porta il cursore del mouse sulla scheda e si preme il tasto si vedrà apparire la stringa "Here!" nel punto in cui è stato fatto clic. Il codice seguente imposta la posizione corrente di disegno alle coordinate in cui l'utente ha premuto il pulsante:

```
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
  Canvas->MoveTo(X, Y);// set pen position
}
```

Ora la pressione del pulsante del mouse imposta la posizione della penna, definendo il punto iniziale della linea. Per disegnare una linea fino al punto in cui l'utente rilascia il pulsante, si deve rispondere ad un evento mouse-up.

Risposta ad un'azione mouse-up

Ogni volta che l'utente rilascia un pulsante del mouse si verifica un evento *OnMouseUp*. Solitamente l'evento va all'oggetto sul quale si trova il cursore del mouse quando l'utente preme il pulsante, che non è necessariamente lo stesso oggetto sul quale si trova il cursore quando il pulsante viene rilasciato. Ciò consente, per esempio, di disegnare una linea come se si estendesse oltre il bordo della scheda.

Per rispondere alle azioni mouse-up, si deve definire un gestore per l'evento *OnMouseUp*.

Il codice seguente è un semplice gestore di evento *OnMouseUp* che disegna una linea fino al punto di rilascio del pulsante del mouse:

```
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button,
```

```
    TShiftState Shift, int X, int Y)
{
    Canvas->LineTo(X, Y);// draw line from PenPos to (X, Y)
}
```

Questo codice permette ad un utente di disegnare linee facendo clic, trascinando e rilasciando. In questo caso, l'utente non può vedere la linea finché il pulsante del mouse non viene rilasciato.

Risposta ad uno spostamento del mouse

Quando l'utente sposta il mouse, si verifica periodicamente un evento *OnMouseMove*. L'evento va all'oggetto che era sotto il puntatore del mouse quando l'utente ha premuto il pulsante. Questo consente di dare all'utente un controllo intermedio disegnando linee temporanee mentre si sposta il mouse.

Per rispondere ai movimenti del mouse, occorre definire un gestore per l'evento *OnMouseMove*. Questo esempio usa gli eventi mouse-move per disegnare figure intermedie su una scheda mentre l'utente tiene premuto il pulsante del mouse, fornendo pertanto all'utente un certo controllo. Il gestore di evento *OnMouseMove* disegna una linea su una scheda nella posizione dell'evento *OnMouseMove*:

```
void __fastcall TForm1::FormMouseMove(TObject *Sender, TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
    Canvas->LineTo(X, Y);// draw line to current position
}
```

Con questo codice, lo spostamento del mouse sulla scheda fa sì che il disegno segua il mouse, anche prima che il pulsante del mouse venga premuto.

Gli eventi mouse-move si verificano quando non sono stati premuti i pulsanti del mouse.

Se si vuole tenere traccia del fatto che sia stato premuto un pulsante del mouse, occorre aggiungere un campo oggetto all'oggetto scheda.

Aggiunta di un campo ad un oggetto scheda per tenere traccia delle azioni del mouse

Per tenere traccia del fatto che sia stato premuto o meno un pulsante del *mouse*, occorre aggiungere un campo oggetto all'oggetto scheda. Quando ad una scheda viene aggiunto un componente, C++Builder aggiunge all'oggetto scheda un campo che rappresenta quel componente, in modo che sia possibile fare riferimento al componente tramite il nome del campo. Inoltre, si possono aggiungere alla scheda campi propri modificando la dichiarazione di tipo nel file header della scheda.

Nell'esempio seguente la scheda ha bisogno di registrare se l'utente ha premuto un pulsante del mouse. Per farlo, essa aggiunge un campo booleana e ne imposta il valore quando l'utente preme il pulsante del mouse.

Per aggiungere un campo ad un oggetto, si deve modificare la definizione di tipo dell'oggetto, specificando l'identificatore di campo e il tipo dopo la direttiva **public** in fondo alla dichiarazione.

CC++Builder “possiede” qualsiasi dichiarazione prima della direttiva **public**: questo è il punto in cui vengono immessi i campi che rappresentano i controlli e i metodi che rispondono agli eventi.

Il codice seguente dà ad una scheda un campo di nome *Drawing* di tipo *bool*, nella dichiarazione dell’oggetto scheda. Inoltre, aggiunge due campi per memorizzare i punti *Origin* e *MovePt* di tipo TPoint.

```
class TForm1 : public TForm
{
__published:  // IDE-managed Components
  void __fastcall FormMouseDown(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y);
  void __fastcall FormMouseMove(TObject *Sender, TShiftState Shift, int X,
  int Y);
  void __fastcall FormMouseUp(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y);
private:// User declarations
public:// User declarations
  __fastcall TForm1(TComponent* Owner);
  bool Drawing;  //field to track whether button was pressed
   TPoint Origin, MovePt;  // fields to store points
};
```

Se si ha un campo *Drawing* che registra le operazioni di disegno, impostarlo a **true** quando si preme il pulsante del mouse e a **false** quando lo si rilascia:

```
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
  Drawing = true;            // set the Drawing flag
  Canvas->MoveTo(X, Y);      // set pen position
}

void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
  Canvas->LineTo(X, Y);      // draw line from PenPos to (X, Y)
  Drawing = false;           // clear the Drawing flag
}
```

Quindi, si può modificare il gestore di evento *OnMouseMove* per disegnare solo quando *Drawing* è **true**:

```
void __fastcall TForm1::FormMouseMove(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
  if (Drawing)
    Canvas->LineTo(X, Y);// only draw if mouse is down
}
```

Questo codice genera un disegno solo tra gli eventi mouse-down e mouse-up, ma genera anche una linea scarabocchiata che traccia gli spostamenti del mouse anziché una linea retta.

Il problema è che ogni volta che il mouse si sposta, il relativo gestore di evento chiama *LineTo*, il quale sposta la posizione della penna, e nel momento in cui viene

rilasciato il pulsante, si perde l'indicazione del punto dal quale doveva partire la linea retta.

Affinamento del disegno delle linee

Con i campi collocati per tracciare i vari punti, è possibile rifinire il disegno di una linea dell'applicazione.

Memorizzazione del punto di origine

Quando si disegnano linee, per tener traccia del punto in cui inizia la linea si utilizza il campo *Origin*.

Origin deve essere impostato al punto in cui si verifica l'evento mouse-down, in modo che il gestore di evento mouse-up possa usare *Origin* per indicare l'inizio della linea, come in questo codice:

```
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
  Drawing = true;            // set the Drawing flag
  Canvas->MoveTo(X, Y);      // set pen position
  Origin = Point(X, Y);      // record where the line starts
}
void __fastcall TForm1::FormMouseUp(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
  Canvas->MoveTo(Origin.x, Origin.y);  // move pen to starting point
  Canvas->LineTo(X, Y);                // draw line from PenPos to (X, Y)
  Drawing = false;                     // clear the Drawing flag
}
```

Queste modifiche servono all'applicazione per disegnare di nuovo la linea finale, e non per disegnare delle azioni intermedie--l'applicazione non supporta ancora l'effetto "elastico."

Momorizzazione dello spostamento

Il problema, almeno per come è scritto al momento il gestore di evento *OnMouseMove*, è che la linea viene disegnata fino alla posizione corrente del mouse, partendo dall'ultima *posizione* del mouse, non da quella originaria. È possibile porre rimedio a questo problema spostando la posizione di disegno al punto di origine, quindi disegnando fino al punto attuale:

```
void __fastcall TForm1::FormMouseMove(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
  if (Drawing)
  {
    Canvas->MoveTo(Origin.x, Origin.y);  // move pen to starting point
    Canvas->LineTo(X, Y);
  }
}
```

Questo codice registra la posizione attuale del mouse, ma le linee intermedie non scompaiono, cosicché risulta difficile vedere la linea finale. L'esempio ha bisogno di eliminare ogni linea prima di disegnare quella successiva, tenendo traccia di dove si trovava quella precedente. Il campo *MovePt* consente di realizzare ciò.

MovePt deve essere impostato al punto finale di ciascuna linea intermedia, in modo che si possa usare *MovePt* e *Origin* per eliminare quella linea la volta successiva in cui viene disegnata una linea:

```
void __fastcall TForm1::FormMouseDown(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
  Drawing = true;            // set the Drawing flag
  Canvas->MoveTo(X, Y);      // set pen position
  Origin = Point(X, Y);      // record where the line starts
  MovePt = Point(X, Y);      // record last endpoint
}
void __fastcall TForm1::FormMouseMove(TObject *Sender, TMouseButton Button,
  TShiftState Shift, int X, int Y)
{
  if (Drawing)
  {
    Canvas->Pen->Mode = pmNotXor;        // use XOR mode to draw/erase
    Canvas->MoveTo(Origin.x, Origin.y);  // move pen to starting point
    Canvas->LineTo(MovePt.x, MovePt.y);  // erase old line
    Canvas->MoveTo(Origin.x, Origin.y);  // move pen to starting point again
    Canvas->LineTo(X, Y);                // draw new line
  }
  MovePt = Point(X, Y);      // record new endpoint
  Canvas->Pen->Mode = pmCopy;
}
```

A questo punto, si ha a disposizione un effetto "elastico" quando si disegna la linea. Cambiando il modo della penna in *pmNotXor*, è possibile combinare la linea con i pixel dello sfondo. Quando si va ad eliminare la linea, in realtà si reimpostano i pixel al modo in cui si trovavano prima. Riportando il modo della penna a *pmCopy* (il suo valore predefinito) dopo aver disegnato le linee, si ha la certezza che la penna è pronta per tracciare il suo disegno finale non appena si rilascia il pulsante del mouse.

Uso di componenti multimediali

C++Builder consente di aggiungere componenti multimediali alle applicazioni Windows (non CLX o per di Linus). A questo scopo, si possono usare il componente *TAnimate* della pagina Win32 o il componente *TMediaPlayer* della pagina System della Component palette. Utilizzare il componente di animazione se si vuole aggiungere all'applicazione un video clip senza l'audio. Utilizzare il componente MediaPlayer, invece, per aggiungere clip audio o video a un'applicazione.

Per ulteriori informazioni sui componenti *TAnimate* e *TMediaPlayer*, consultare la Guida in linea della VCL.

In questa sezione vengono trattati i seguenti argomenti:

- Aggiunta di clip video senza suono ad un'applicazione
- Aggiunta di clip audio e/o video ad un'applicazione

Aggiunta di clip video senza suono ad un'applicazione

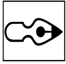
Animation in C++Builder consente di aggiungere all'applicazione clip video senza suono.

Per aggiungere un clip video senza suono ad un'applicazione:

1. Fare doppio clic sull'icona Animate sulla pagina Win32 della Component palette. Questa operazione inserisce automaticamente un controllo di animazione nella finestra della scheda in cui si intende visualizzare il clip video.
2. Usando l'Object Inspector, selezionare la proprietà *Name* ed immettere un nuovo nome per il controllo di animazione. Questo nome verrà usato quando si chiama il controllo di animazione. (Per attribuire un nome agli identificatori C++, occorre seguire le regole standard).

   Per impostare in fase di progettazione le proprietà e per creare i gestori di evento, si deve lavorare sempre direttamente nell'Object Inspector.
3. Eseguire una delle seguenti operazioni:
   - Selezionare la proprietà *Common* AVI e scegliere una delle AVI disponibili dall'elenco a discesa; oppure
   - Selezionare la proprietà *FileName* e fare clic sul pulsante ellissi (…), scegliere un file AVI da una directory locale o di rete disponibile e fare clic su *Open* nella finestra di dialogo Open AVI; oppure
   - Selezionare la risorsa di un file AVI attraverso le proprietà *ResName* o *ResID*. Per indicare il modulo che contiene la risorsa identificata da *ResName* o *ResID*, si deve usare *ResHandle*.

   Tale operazione carica il file AVI in memoria. Se si desidera visualizzare il primo frame del clip *AVI* sullo schermo finché non viene riprodotto usando la proprietà *Active* o il metodo *Play*, impostare la proprietà *Open* a **true**.
4. Impostare la proprietà *Repetitions* al numero di volte che il clip AVI deve essere riprodotto. Se questo valore è 0, la sequenza viene ripetuta finché non viene chiamato il metodo *Stop*.
5. Apportare tutte le altre modifiche alle impostazioni del controllo di animazione. Per esempio, se si desidera cambiare il primo frame visualizzato quando si apre il controllo di animazione, impostare la proprietà *StartFrame* al valore del frame desiderato.
6. Impostare la proprietà *Active* a **true** usando l'elenco a discesa o scrivere un gestore di evento per eseguire il clip AVI quando in fase di esecuzione si verifica uno specifico evento. Per esempio, per attivare il clip AVI quando si fa clic su un oggetto pulsante, si deve scrivere l'evento *OnClick* del pulsante che lo specifica. In alternativa, per specificare quando deve essere riprodotto l'AVI si può chiamare il metodo *Play*.



Se vengono apportate modifiche alla scheda o ad un qualsiasi suo componente dopo aver impostato *Active* a **true**, la proprietà *Active* diventa **false** ed è necessario reimpostarla a **true**. Questa operazione deve essere effettuata prima di avviare l'applicazione o durante la sua esecuzione.

Esempio di aggiunta di clip video senza suono

Si supponga di voler visualizzare un logo animato come prima schermata di un'applicazione. Terminata la riproduzione del logo, la schermata deve scomparire.

Per eseguire questo esempio, occorre creare un nuovo progetto e salvare il file Unit1.cpp come Frmlogo.cpp e il file Project1.cpp come Logo.bpr. Quindi:

1. Fare doppio clic sull'icona del controllo di animazione sulla pagina Win32 della Component palette.
2. Usando l'Object Inspector, impostarne la proprietà *Name* a *Logo1*.
3. Selezionare la proprietà FileName, fare clic sul pulsante ellissi (…), scegliere il file dillo.avi dalla directory ..\Examples\MFC\General|Cmnctrls. Quindi fare clic su Open nella finestra di dialogo Open AVI.

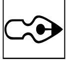
   Questa operazione carica in memoria il file dillo.avi.
4. Posizionare sulla scheda la casella del controllo di animazione, facendo clic e trascinandola nell'angolo superiore destro della scheda.
5. Impostarne la proprietà Repetitions a 5.
6. Fare clic sulla scheda per assegnarle il fuoco, e impostarne la proprietà *Name* a *LogoForm1* e la proprietà Caption a *Logo Window*. A questo punto diminuire l'altezza della scheda per centrare a destra il controllo Animate.
7. Fare doppio clic sull'evento *OnActivate* della scheda e scrivere il seguente codice per eseguire il clip AVI quando la scheda ha il fuoco in fase di esecuzione:

   ```
   Logo1->Active = true;
   ```
8. Fare doppio clic sull'icona Label nella pagina Standard della Component palette. Selezionare la proprietà Caption e immettere *Welcome to Armadillo Enterprises 4.0*. Quindi, selezionare la proprietà Font, fare clic sul pulsante ellissi (…), quindi scegliere nella finestra di dialogo Font come Font Style: Bold, Size: 18, Color: Navy e fare clic su OK. Fare clic e trascinare il controllo etichetta per centrarlo nella scheda.
9. Fare clic sul controllo di animazione per ridargli il fuoco. Fare doppio clic sul suo evento *OnStop* e scrivere il seguente codice per chiudere la scheda quando il file AVI termina l'esecuzione:

   ```
   LogoForm1->Close();
   ```
10. Selezionare Run|Run per eseguire la finestra del logo animato.

Aggiunta di clip audio e/o video a un'applicazione

Il componente MediaPlayer di C++Builder consente di aggiungere audio e/o video clip all'applicazione. Questo componente apre un dispositivo mediale e riproduce, termina, sospende, registra, ecc., i clip audio e/o video usati dal dispositivo mediale. Questo può essere un dispositivo hardware o software.



Nella programmazione multipiattaforma il supporto per clip audio e video non è disponibile.

Per aggiungere a un'applicazione un clip audio e/o video:

1 Fare doppio clic sull'icona MediaPlayer nella pagina System della Component palette. Questa operazione colloca automaticamente un controllo MediaPlayer nella finestra della scheda nella quale si vuole la funzione mediale.

2 Usando l'Object Inspector, selezionare la proprietà *Name* e immettere un nuovo nome per il controllo MediaPlayer. Questa procedura deve essere eseguita quando si chiama il controllo MediaPlayer. (Per dare un nome agli identificatori C++, si devono usare le regole standardntifiers.)

   Per impostare in fase di progettazione le proprietà e per creare i gestori di evento, si deve lavorare sempre direttamente nell'Object Inspector.

3 Selezionare la proprietà *DeviceType* e scegliere il tipo di dispositivo appropriato da aprire usando la proprietà *AutoOpen* o il metodo *Open*. (Se *DeviceType* è dtAutoSelect, il tipo di dispositivo viene selezionato in base all'estensione del file mediale specificato dalla proprietà *FileName*.) Per ulteriori informazioni sui tipi di dispositivo e sulle loro funzioni, consultare laTabella 10.6.

4 Se il dispositivo memorizza il suo supporto in un file, specificarne il nome usando la proprietà *FileName*. Selezionare la proprietà *FileName*, fare clic sul pulsante coi puntini di sospensione (…) e scegliere un file mediale da una directory locale o di rete disponibile e fare clic su *Open* nella finestra di dialogo omonima. In alternativa, inserire in fase di esecuzione l'hardware (disco, cassetta o altro) in cui è memorizzato il supporto per il dispositivo mediale selezionato.

5 Impostare la proprietà *AutoOpen* a **true**. In questo modo il riproduttore mediale apre automaticamente il dispositivo specificato quando viene creata in fase di esecuzione la scheda che contiene il controllo MediaPlayer.Se *AutoOpen* è **false**, il dispositivo deve essere aperto con una chiamata al metodo *Open*.

6 Impostare la proprietà *AutoEnable* a **true** per attivare o disattivare automaticamente i pulsanti MediaPlayer, come richiesto in fase di esecuzione; oppure, fare doppio clic sulla proprietà *EnabledButtons* per impostare ciascun pulsante a **true** o **false**, a seconda di quali si desidera attivare o disattivare.

   Il dispositivo multimediale viene riprodotto, messo in pausa, fermato, e così via, quando l'utente fa clic sul pulsante corrispondente sul componente Mediaplayer. Il dispositivo può essere controllato anche tramite i metodi che corrispondono ai pulsanti (Play, Pause, Stop, Next, Previous, e così via).

7 Posizionare la barra del controllo MediaPlayer sulla scheda facendo clic e trascinandola nel posto desiderato sulla scheda o selezionando la proprietà *Align* e scegliendo la posizione di allineamento corretta dall'elenco a discesa.

   Se si vuole che il MediaPlayer sia invisibile in fase di esecuzione, impostare la proprietà *Visible* a **false** e controllare il dispositivo chiamando i metodi appropriati (*Play*, *Pause*, *Stop*, *Next*, *Previous*, *Step*, *Back*, *Start Recording*, *Eject*).

8 Apportare tutte le altre modifiche alle impostazioni del controllo MediaPlayer. Per esempio, se il supporto richiede una finestra di visualizzazione, impostare la proprietà *Display* al controllo che visualizza il supporto. Se il dispositivo usa più tracce, impostare la proprietà *Tracks* alla traccia desiderata.

Tabella 10.6 Tipi di dispositivi multimediali e loro funzioni

Tipo di dispositivo	Software/Hardware usato	Esegue	Usa le tracce	Usa una finestra di visualizzazione
dtAVIVideo	AVI Video Player for Windows	File Video *AVI*	No	Sì
dtCDAudio	CD Audio Player for Windows or a CD Audio Player	CD Audio	Sì	No
dtDAT	Digital Audio Tape Player	Nastri audio digitali	Sì	No
dtDigitalVideo	Digital Video Player for Windows	File *AVI*, MPG, MOV	No	Sì
dtMMMovie	MM Movie Player	Film MM	No	Sì
dtOverlay	Overlay device	Video analogici	No	Sì
dtScanner	Image Scanner	N/D per *Play* (Record fa scorrere le immagini)	No	No
dtSequencer	MIDI Sequencer for Windows	MIDI files	Sì	No
dtVCR	Video Cassette Recorder	Videocassette	No	Sì
dtWaveAudio	Wave Audio Player for Windows	File WAV	No	No

Esempio di aggiunta di clip audio e/o video (solo VCL)

Questo esempio esegue un video clip AVI di un'inserzione multimediale per C++Builder. Per eseguire questo esempio, occorre creare un nuovo progetto e salvare il file Unit1.cpp in FrmAd.cpp e il file Project1.bpr in MmediaAd.bpr. Quindi:

1 Fare doppio clic sull'icona MediaPlayer nella pagina System della Component palette.

2 Usando l'Object Inspector, impostare la proprietà *Name* del MediaPlayer a *VideoPlayer1*.

3 Selezionare la sua proprietà DeviceType e scegliere dtAVIVideo dall'elenco a discesa.

4 Selezionare la proprietà FileName, fare clic sul pulsante coi puntini di sospensione (…), scegliere il file dalla directory ..\Examples\Coolstuf.directory. Fare clic su Open nella finestra di dialogo omonima.

5 Impostare la proprietà AutoOpen a **true** e la proprietà Visible a **false**.

6 Fare doppio clic sull'icona Animate dalla pagina Win32 della Component palette. Impostarne la proprietà AutoSize a **false**, la proprietà Height a *175* e la proprietà *Width* a *200*. Fare clic e trascinare il controllo di animazione nell'angolo superiore sinistro della scheda.

7 Fare clic sul MediaPlayer per ridargli il fuoco. Selezionare la sua proprietà Display e scegliere Animate1 dall'elenco a discesa.

8 Fare clic sulla scheda per darle il fuoco e selezionare la sua proprietà *Name* e immettere *C++_Ad*. A questo punto, riportare la scheda alle dimensioni del controllo di animazione.

9 Fare doppio clic sull'evento *OnActivate* della scheda e scrivere il seguente codice per eseguire il video *AVI* quando la scheda ha il fuoco:

```
VideoPlayer1->Play();
```

10 Scegliere Run|Run per eseguire il video AVI.

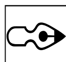
Capitolo

11

Scrittura di applicazioni multi-thread

C++Builder fornisce diversi oggetti che semplificano la scrittura di applicazioni multi-thread. Queste sono applicazioni che includono diversi percorsi simultanei di esecuzione. Poiché l'uso di più thread comporta una certa complessità, è possibile migliorare i programmi:

- **Evitando i colli di bottiglia**. Con un solo thread un programma deve bloccare completamente l'esecuzione quando aspetta che terminino processi lenti come l'accesso ai file sul disco, la comunicazione con altre macchine o la visualizzazione di un contenuto multimediale. La CPU rimane in stato di attesa finché il processo non viene completato. Con più thread l'applicazione può proseguire l'esecuzione in thread diversi mentre un thread aspetta i risultati di un processo lento.
- **Organizzando il comportamento del programma**. Spesso il comportamento di un programma può essere organizzato in più processi paralleli che funzionano in modo indipendente. Usare i thread per lanciare una singola sezione di codice contemporaneamente per ciascuno di questi casi paralleli. Usare i thread per assegnare la priorità alle varie operazioni del programma in modo che la CPU possa avere più tempo a disposizione per le operazioni più importanti.
- **Con l'elaborazione multiprocesso**. Se il sistema che esegue il programma ha più processori, è possibile migliorare le prestazioni dividendo il lavoro in diversi thread e permettendo l'esecuzione simultanea su processori distinti.

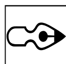


Non tutti i sistemi operativi implementano una vera elaborazione multiprocesso, anche nel caso sia supportata dall'hardware utilizzato. Ad esempio, Windows 9x simula solamente l'elaborazione multiprocesso, anche se l'hardware utilizzato lo consentirebbe.

Definizione di oggetti thread

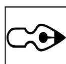
Per la maggior parte delle applicazioni è possibile usare un oggetto thread per rappresentare un thread di esecuzione di un'applicazione. Gli oggetti thread

semplificano la scrittura di applicazioni multi-thread incapsulando i thread che è necessario usare più frequentemente.



Gli oggetti thread non consentono di controllare gli attributi di sicurezza o le dimensioni dello stack dei thread. Qualora questo controllo sia necessario, occorre usare le funzioni *CreateThread o BeginThread* delle API di Windows. Anche quando si usano le Windows Thread API calls o *BeginThread*, è sempre possibile sfruttare i vantaggi di alcuni degli oggetti thread di sincronizzazione e dei metodi descritti nella sezione “Coordinamento dei thread” a pagina 11-7. Per maggiori informazioni sull’utilizzo di *CreateThread o BeginThread*, consultare la Guida in linea di Windows.

Per usare un oggetto thread in un’applicazione, è necessario creare un nuovo discendente di *TThread*. A questo scopo, scegliere File|New dal menu principale. Nella finestra di dialogo New Items, fare doppio clic su Thread-Object (oppure su Thread Object di CLX per applicazioni multipiattaforma) e immettere un nome di classe, come *TMyThread*. Per assegnare un nome a questo nuovo thread, selezionare la casella di controllo Named Thread e immettere un nome per il thread (solo per VCL). L’assegnazione di un nome al thread semplifica il tracciamento del thread durante il debug. Una volta fatto clic su OK, C++Builder crea un nuovo file .cpp e un file header per implementare il thread. Per ulteriori informazioni sull’assegnazione di un nome per il thread, si veda il paragrafo “Assegnazione di un nome al thread” a pagina 11-13.



A differenza della maggior parte delle finestre di dialogo nell’IDE che richiedono un nome di classe, la finestra di dialogo New Thread Object non mette automaticamente una ‘T’ come prefisso del nome di classe fornito.

Il file .cpp generato automaticamente contiene lo schema di codice per il nuovo oggetto thread. Un thread al quale è stato attribuito come nome *TMyThread* apparirà più o meno in questo modo:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Unit2.h"
#pragma package(smart_init)
//---------------------------------------------------------------------------
__fastcall TMyThread::TMyThread(bool CreateSuspended): TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
void __fastcall TMyThread::Execute()
{
// ---- Place thread code here ----
}
//---------------------------------------------------------------------------
```

Occorre completare il codice per il costruttore e per il metodo *Execute*. Le fasi relative a tale operazione vengono descritte nelle sezioni successive.

Inizializzazione del thread

Utilizzare il costruttore per inizializzare una nuova classe thread. È qui che si può assegnare una priorità predefinita per il thread e indicare se verrà rilasciato automaticamente al termine dell'esecuzione.

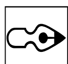
Assegnazione di una priorità predefinita

La priorità indica il grado di attenzione che il thread riceve quando il sistema operativo programma il tempo che la CPU deve dedicare ad ogni thread dell'applicazione. Per gestire operazioni per le quali il tempo è determinante, si deve usare un thread ad alta priorità, mentre per tutte le altre operazioni si può ricorrere ad un thread a bassa priorità. Per indicare la priorità dell'oggetto thread, impostare la proprietà *Priority*.

Se si sta scrivendo un'applicazione Windows, i valori di *Priority* sono disposti su una scala, come descritto nella Tabella 11.1:

Tabella 11.1 Priorità del thread

Valori	Priorità
tpIdle	Il thread viene eseguito solo quando il sistema è inattivo. Windows non interromperà gli altri thread per eseguirne uno con priorità *tpIdle*.
tpLowest	La priorità del thread è due punti sotto il normale.
tpLower	La priorità del thread è un punto sotto il normale.
tpNormal	Il thread ha priorità normale.
tpHigher	La priorità del thread è un punto sopra il normale.
tpHighest	La priorità del thread è due punti sopra il normale.
tpTimeCritical	Il thread riceve la priorità più alta.



Se si scrive un'applicazione multipiattaforma, è necessario utilizzare codice differente per l'assegnazione delle priorità in Windows e Linux. In Linux, *Priority* è un valore numerico che dipende dalle politiche di threading che puossono essere cambiate solo dall'utente root. Per i dettagli, consultare la versione CLX di *TThread* e *Priority* nella Guida in linea.



L'aumento della priorità del thread di un'operazione che richiede un uso intensivo della CPU può lasciare "a bocca asciutta" gli altri thread dell'applicazione. Esso si applica solo ai thread che trascorrono la maggior parte del tempo in attesa di eventi esterni.

Il codice seguente mostra il costruttore di un thread a bassa priorità che esegue operazioni secondarie che non interferiranno con le altre prestazioni dell'applicazione:

```
//---------------------------------------------------------------------------
__fastcall TMyThread::TMyThread(bool CreateSuspended): TThread(CreateSuspended)
{
  Priority = tpIdle;
}

//---------------------------------------------------------------------------
```

Come indicare quando i thread vengono rilasciati

Quando i thread terminano le rispettive operazioni, di solito vengono semplicemente rilasciati. In questo caso, è più facile permettere all'oggetto thread di rilasciarsi da solo. Per fare ciò, impostare la proprietà *FreeOnTerminate* a **true**.

A volte, tuttavia, il termine di un thread deve essere coordinato con altri thread. Ad esempio, potrebbe essere necessario aspettare il valore restituito da un thread prima di eseguire un'azione in un altro thread. In questo caso, non si potrà rilasciare il primo thread finché il secondo non ha ricevuto il valore restituito. Questa situazione può essere gestita impostando *FreeOnTerminate* a **false** e quindi rilasciare il primo thread in modo esplicito dall'interno del secondo.

Scrittura della funzione thread

Il metodo *Execute* è una funzione thread. Può essere considerato come un programma che viene lanciato dall'applicazione, con la differenza che condivide lo stesso spazio di elaborazione. La scrittura della funzione thread è un po' più difficile della scrittura di un programma distinto perché richiede la verifica del fatto che non venga sovrascritta la memoria usata dagli altri thread dell'applicazione. D'altra parte, dal momento che il singolo thread condivide lo stesso spazio di elaborazione con gli altri thread, è possibile usare la memoria condivisa per la comunicazione tra i thread.

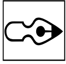
Utilizzo del thread VCL/CLX principale

Quando si usano oggetti della gerarchia di oggetti della VCL o di CLX, non si ha alcuna garanzia sul fatto che le loro proprietà e i loro metodi salvaguardino i thread. Ossia, l'accesso alle proprietà o l'esecuzione dei metodi possono eseguire operazioni che usano memoria non protetta dalle operazioni degli altri thread. Tenuto conto di ciò, si imposta un thread principale per l'accesso agli oggetti della VCL e di CLX. Questo è il thread che gestisce tutti i messaggi di Windows ricevuti dai componenti dell'applicazione.

Se tutti gli oggetti accedono alle loro proprietà ed eseguono i loro metodi all'interno di questo singolo thread, non occorre preoccuparsi del fatto che interferiscano tra loro. Per usare il thread principale, creare una routine distinta che esegua le operazioni desiderate. Richiamarla quindi dall'interno del metodo *Synchronize* del thread. Per esempio:

```
void __fastcall TMyThread::PushTheButton(void)
{
  Button1->Click();
}
⋮
void __fastcall TMyThread::Execute()
{
  ⋮
  Synchronize((TThreadMethod)PushTheButton);
  ⋮
}
```

Synchronize aspetta il thread principale per avviare il ciclo di messaggi, dopo di che esegue il metodo passato.



Dal momento che *Synchronize* usa il ciclo di messaggi, non funziona nelle applicazioni console. Per proteggere l'accesso agli oggetti VCL e CLX nelle applicazioni per console, si devono, pertanto, usare altri meccanismi, come le sezioni critiche.

Non occorre usare sempre il thread principale. Alcuni oggetti sono associati ai thread. Omettendo l'uso del metodo *Synchronize* quando si è certi che i metodi di un oggetto sono thread-safe si miglioreranno le prestazioni, poiché non occorrerà aspettare che il thread VCL o CLX avvii il proprio ciclo di messaggi. Non è necessario usare il metodo *Synchronize* nelle seguenti situazioni:

- I componenti per l'accesso ai dati sono thread-safe a patto che, per i dataset che utilizzano BDE, ciascun thread abbia il proprio componente session di database. L'unica eccezione si presenta quando si utilizzano i driver di Access Microsoft, che sono costruiti usando una libreria Microsoft, che non è thread-safe. Per dbDirect, finché la libreria client del produttore è thread-safe, i componenti dbDirect saranno thread-safe. I componenti ADO e InterbaseExpress sono thread-safe.

  Quando si usano componenti di accesso ai dati, occorre sempre passare al metodo *Synchronize* tutte le chiamate che interessano i controlli associati ai dati. Quindi, per esempio, occorre sincronizzare le chiamate che collegano un controllo dati ad un dataset impostando la proprietà *DataSet* dell'oggetto data source, ma non occorre alcuna sincronizzazione per accedere ai dati di un campo del dataset.

  Per maggiori informazioni sull'utilizzo delle sessioni database con i thread, in applicazioni che utilizzano BDE, consultare "Gestione di sessioni multiple" a pagina 24-29.
- Gli oggetti DataCLX sono thread-safe mentre gli oggetti VisualCLX non lo sono.
- Gli oggetti grafici sono thread-safe. Non occorre usare il thread VCL principale per accedere a *TFont*, *TPen*, *TBrush*, *TBitmap*, *TMetafile* (solo VCL), *TDrawing*(solo CLX), o *TIcon*. Gli oggetti canvas possono essere utilizzati all'esterno del metodo *Synchronize* bloccandoli (vedere "Blocco di oggetti" a pagina 11-8).
- Benché gli oggetti list non siano thread-safe, si può usare una versione thread-safe, *TThreadList*, al posto di *TList*.

Chiamare periodicamente la routine *CheckSynchronize* dall'interno del thread principale dell'applicazione in modo che i thread in background possano sincronizzare la propria esecuzione con il thread principale. Il momento ideale per chiamare *CheckSynchronize* è quando l'applicazione è inattiva (ad esempio, da un gestore di evento *OnIdle*). Ciò garantisce che è sicuro effettuare chiamate ai metodi nel thread di background.

Uso di variabili locali al thread

Il metodo *Execute* e tutte le routine che chiama hanno proprie variabili locali, esattamente come qualsiasi altra routine di C++. Queste routine possono accedere anche a tutte le variabili globali. Infatti, queste variabili forniscono un potente meccanismo per la comunicazione tra i thread.

Qualche volta, tuttavia, può risultare preferibile usare variabili che sono globali per tutte le routine che vengono eseguite nel thread, ma non sono condivise con altre istanze della stessa classe thread. Per ottenere questo risultato, occorre dichiarare le variabili locali al thread. Creare una variabile locale al thread aggiungendo alla dichiarazione della variabile il modificatore **__thread**. Ad esempio:

```
int __thread x;
```

dichiara una variabile di tipo integer che è privata per ciascun thread dell'applicazione, ma globale all'interno di ciascun thread.

Il modificatore **__thread** può essere utilizzato solamente per variabili globali (il cui campo d'azione è il file) e statiche. Pointer e Function non possono essere variabili thread. Non funzionano come variabili neppure i tipi che usano la semantica a sola scrittura, così come pure le variabili di tipo AnsiStrings. Un elemento di programma che richieda una inizializzazione o una finalizzazione al momento dell'esecuzione non può essere dichiarato di tipo **__thread**.

Le seguenti dichiarazioni richiedono una inizializzazione al momento dell'esecuzione e pertanto non sono consentite.

```
int f( );
int __thread x = f( );   // illegal
```

La creazione di un'istanza di una classe mediante un costruttore o un distruttore definito dall'utente richiede una inizializzazione al momento dell'esecuzione e non è, pertanto, consentita:

```
class X  {
   X( );
   ~X( );
};
X __thread myclass;   // illegal
```

Controllo del termine da parte di altri thread

Il thread inizia l'esecuzione quando viene chiamato il metodo *Execute* (vedere "Esecuzione di oggetti thread" a pagina 11-12) e continua finché *Execute* non finisce. Ciò riflette il modello che prevede che il thread esegua una specifica operazione e quindi si fermi non appena ha terminato. Qualche volta, tuttavia, un'applicazione ha bisogno di un thread da eseguire finché non viene soddisfatto un criterio esterno.

Controllando la proprietà *Terminated*, si può consentire ad altri thread di segnalare al thread attivo che è il momento di terminare l'esecuzione. Quando un altro thread tenta di interrompere il thread attivo, chiama il metodo *Terminate*. *Terminate* imposta la proprietà *Terminated* del thread attivo a **true**. Spetta al metodo *Execute* implementare il metodo *Terminate* controllando e rispondendo alla proprietà *Terminated*. Il seguente esempio mostra un modo per eseguire questa operazione:

```
void __fastcall TMyThread::Execute()
{
  while (!Terminated)
    PerformSomeTask();
}
```

Gestione delle eccezioni nella funzione di thread

Il metodo *Execute* deve intercettare tutte le eccezioni che si verificano nel thread. Se non si riesce a intercettare un'eccezione nella funzione di thread, l'applicazione potrebbe provocare violazioni di accesso. Ciò potrebbe non essere evidente quando si sta sviluppando l'applicazione, in quanto l'IDE intercetta le eccezioni, ma quando si eseguirà l'applicazione esternamente al debugger, l'eccezione causerà un errore di esecuzione e l'applicazione si bloccherà.

Per intercettare le eccezioni che si verificano all'interno della funzione di thread, aggiungere un blocco **try**...**catch** all'implementazione del metodo *Execute*:

```
void __fastcall TMyThread::Execute()
{
  try
  {
    while (!Terminated)
      PerformSomeTask();
  }
  catch (...)
  {
    // do something with exceptions
  }
}
```

Scrittura del codice di pulizia

È possibile centralizzare il codice di pulizia quando il thread termina l'esecuzione. Appena prima che un thread cessi l'esecuzione, si verifica un evento *OnTerminate*. Per garantire che venga sempre eseguito, indipendentemente dal percorso di esecuzione seguito dal metodo *Execute*, bisogna collocare il codice di pulizia nel gestore di eventi *OnTerminate*.

Il gestore di eventi *OnTerminate* non viene eseguito come parte del thread. Viene invece eseguito nel contesto del thread VCL o CLX principale dell'applicazione. Ciò comporta due implicazioni:

- Non è possibile usare nessuna delle variabili locali al thread in un gestore di eventi *OnTerminate* (a meno che non si vogliano i valori del thread VCL o CLX principale).
- Si può accedere con tranquillità a tutti i componenti e agli oggetti VCL o CLX dal gestore di eventi *OnTerminate* senza preoccuparsi dell'eventuale conflitto con altri thread.

Per maggiori informazioni sul thread principale di VCL o di CLX, vedere "Utilizzo del thread VCL/CLX principale" a pagina 11-4.

Coordinamento dei thread

Quando si scrive del codice da eseguire quando viene attivato il thread, bisogna considerare il comportamento degli altri thread che possono essere eseguiti

contemporaneamente. In particolare, si deve prestare molta attenzione ad evitare che due thread cerchino di usare lo stesso oggetto o la stessa variabile globale nello stesso istante. Inoltre, il codice di un thread può dipendere dai risultati di operazioni eseguite da altri thread.

Come evitare l'accesso simultaneo

Per evitare conflitti con altri thread quando si accede agli oggetti o alle variabili globali, può risultare necessario bloccare l'esecuzione degli altri thread finché il codice del thread non ha terminato un'operazione. Si deve fare attenzione a non bloccare l'esecuzione di altri thread senza che ce ne sia la necessità. Ciò potrebbe provocare un serio degrado delle prestazioni e annullare quasi tutti i vantaggi derivanti dall'uso di più thread.

Blocco di oggetti

Alcuni oggetti hanno un blocco integrato che impedisce l'esecuzione di altri thread per l'uso dell'istanza di quell'oggetto.

Per esempio, gli oggetti canvas (*TCanvas* e discendenti) hanno un metodo *Lock* che impedisce agli altri thread di accedere al canvas finché non viene chiamato il metodo *Unlock*.

Anche la VCL e la CLX includono un oggetto list thread-safe, *TThreadList*. Chiamando *TThreadList::LockList* viene restituito l'oggetto list e viene anche impedito ad altri thread in esecuzione di usare la lista finché non viene chiamato il metodo *UnlockList*. Le chiamate a *TCanvas::Lock* o a *TThreadList::LockList* possono essere annidate tranquillamente. Il blocco non viene rilasciato finché non si verifica una corrispondenza tra l'ultima chiamata di blocco ed una chiamata di sblocco nello stesso thread.

Uso di sezioni critiche

Se gli oggetti non forniscono un blocco integrato, si può usare una sezione critica. Queste sezioni funzionano come cancelli che consentono l'ingresso di un solo thread per volta. Per usare una sezione critica, creare un'istanza globale di *TCriticalSection. TCriticalSection* ha due metodi, *Acquire* (che impedisce agli altri thread di eseguire la sezione) e *Release* (che rimuove il blocco).

Ciascuna sezione critica è associata alla memoria globale che si intende proteggere. Ogni thread che accede a quella memoria globale deve usare prima il metodo *Acquire* per verificare che nessun altro thread la stia usando. Alla fine, i thread chiamano il metodo *Release* in modo che altri thread possano accedere alla memoria globale chiamando *Acquire*.



Le sezioni critiche funzionano solo se ogni thread le usa per accedere alla memoria globale associata. I thread che ignorano la sezione critica e accedono alla memoria globale senza chiamare *Acquire* possono provocare problemi di accesso simultaneo.

Per esempio, si consideri un'applicazione che ha una variabile globale della sezione critica, *pLockXY*, che blocca l'accesso alle variabili globali X e Y. Qualsiasi thread che

usa X o Y deve evitare di utilizzarla con chiamate alla sezione critica come nell'esempio seguente:

```
pLockXY->Acquire(); // lock out other threads
try
{
  Y = sin(X);
}
__finally
{
  pLockXY->Release();
}
```

Uso del sincronizzatore multilettura a scrittura esclusiva

Quando si usano sezioni critiche per proteggere la memoria globale, solo un thread per volta può usare la memoria. Ciò può creare un livello di protezione superiore a quello richiesto, specialmente se si ha a che fare con un oggetto o una variabile che devono essere letti spesso ma su cui si scrive molto raramente. Non si corre alcun rischio se diversi thread leggono contemporaneamente la stessa memoria, finché nessun thread cerca di scrivervi sopra.

Quando si ha a che fare con della memoria globale che viene letta spesso, ma nella quale i thread scrivono raramente, si può proteggerla usando *TMultiReadExclusiveWriteSynchronizer*. Questo oggetto si comporta come una sezione critica, che consente però a diversi thread di leggere la memoria che protegge, sempre che non vi sia un altro thread che vi sta scrivendo. I thread devono avere accesso esclusivo per la scrittura nella memoria protetta tramite *TMultiReadExclusiveWriteSynchronizer*.

Per usare un sincronizzatore multilettura a scrittura esclusiva, occorre creare un'istanza globale di *TMultiReadExclusiveWriteSynchronizer* che viene associata alla memoria globale che si intende proteggere. Ogni thread che legge da questa memoria chiama prima il metodo *BeginRead*. *BeginRead* garantisce che nessun altro thread stia al momento scrivendo nella memoria. Quando un thread termina la lettura della memoria protetta, chiama il metodo *EndRead*. Tutti i thread che scrivono nella memoria protetta devono chiamare prima *BeginWrite*. *BeginWrite* garantisce che nessun altro thread stia al momento leggendo o scrivendo nella memoria. Quando un thread finisce di scrivere nella memoria protetta, chiama il metodo *EndWrite*, in modo che i thread che sono in attesa di leggere la memoria possano iniziare.



Come le sezioni critiche, il sincronizzatore multilettura a scrittura esclusiva funziona solo se viene utilizzato da tutti i thread per accedere alla memoria globale associata.I thread che ignorano il sincronizzatore ed accedono alla memoria globale senza chiamare *BeginRead* o *BeginWrite* creano problemi di accesso simultaneo.

Altre tecniche per la condivisione della memoria

Quando si utilizzano oggetti della gerarchia della VCL o CLX, per eseguire il codice si deve ricorrere al thread principale. L'uso di questo thread assicura che l'oggetto non acceda indirettamente alla memoria che viene usata anche dagli oggetti VCL o CLX in altri thread. L'utilizzo di questo thread assicura che l'oggetto non acceda indirettamente alla memoria che viene usata anche dagli oggetti di VCL e di CLX in

altri thread. Per maggiori informazioni sul thread principale, vedere “Utilizzo del thread VCL/CLX principale” a pagina 11-4.

Se la memoria globale non ha bisogno di essere condivisa tra più thread, bisogna considerare la possibilità di usare variabili locali al thread al posto delle variabili globali. Usando variabili locali al thread, il thread non ha bisogno di aspettare o di sbloccare un altro thread. Per maggiori informazioni sulle variabili locali al thread, vedere “Uso di variabili locali al thread” a pagina 11-5.

Attesa di altri thread

Se il thread utilizzato deve attenderne un altro per finire una certa operazione, è possibile comunicargli di sospendere temporaneamente l’esecuzione. Si può aspettare che un altro thread termini completamente l’esecuzione oppure che segnali che ha completato un’operazione.

Attesa che un thread termini l’esecuzione

Per aspettare che un altro thread finisca l’esecuzione, usare il metodo *WaitFor* dell’altro thread. *WaitFor* non restituisce il controllo finché l’altro thread non interrompe l’esecuzione, completando il suo metodo Executeo terminando a causa di un’eccezione. Per esempio, il codice seguente aspetta finché un altro thread non riempie un oggetto list del thread, prima di accedere agli oggetti della lista:

```
if (pListFillingThread->WaitFor())
{
  TList *pList = ThreadList1->LockList();
  for (int i = 0; i < pList->Count; i++)
    ProcessItem(pList->Items[i]);
  ThreadList1->UnlockList();
}
```

Nell’esempio precedente gli elementi della lista erano accessibili solo quando il metodo *WaitFor* indicava che la lista era stato effettivamente riempita. Questo valore restituito deve essere assegnato dal metodo *Execute* del thread che era atteso. Tuttavia, poiché i thread che chiamano *WaitFor* vogliono conoscere il risultato dell’esecuzione del thread e non il codice che chiama *Execute*, il metodo *Execute* non restituisce alcun valore. Invece, il metodo *Execute* imposta la proprietà *ReturnValue*. *ReturnValue* viene quindi restituito dal metodo WaitForquando viene chiamato da altri thread. I valori restituiti sono interi. L’applicazione determina il loro significato.

Attesa che venga completata un’operazione

Qualche volta è necessario aspettare che un thread finisca una certa operazione piuttosto che attendere che uno specifico thread termini l’esecuzione. Per ottenere questo risultato, occorre usare un oggetto evento. Gli oggetti evento (*TEvent*) devono essere creati con un campo d’azione globale in modo che possano agire come segnali visibili per tutti i thread.

Quando un thread completa un’operazione da cui dipendono altri thread, chiama *TEvent::SetEvent*. *SetEvent* attiva il segnale, in modo che tutti gli altri thread che

controllano sapranno che l'operazione è stata completata. Per disattivare il segnale, usare il metodo *ResetEvent*.

Per esempio, si consideri una situazione in cui si deve aspettare che terminino l'esecuzione diversi thread anziché uno solo. Dal momento che non si sa quale thread finirà per ultimo, non basta usare il metodo *WaitFor* di uno dei thread. Occorre, invece, che ogni thread incrementi un contatore quando ha finito, e che l'ultimo thread segnali la fine dell'intera operazione attivando un evento. Instead, you can have each thread increment a counter when it is finished, and have the last thread signal that they are all done by setting an event.

Il codice seguente mostra la fine del gestore di evento *OnTerminate* per tutti i thread che devono completare l'esecuzione. *CounterGuard* è un oggetto sezione critica globale, che impedisce che più thread utilizzino il contatore contemporaneamente. *Counter* è una variabile globale che conta il numero di thread che sono stati completati.

```
void __fastcall TDataModule::TaskThreadTerminate(TObject *Sender)
{
  ⋮
  CounterGuard->Acquire(); // lock the counter
  if (--Counter == 0)    // decrement the global counter
    Event1->SetEvent(); // signal if this is the last thread
  CounterGuard->Release(); // release the lock on the counter
}
```

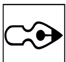
Il thread principale inizializza la variabile *Counter*, avvia l'esecuzione dei thread e aspetta il segnale che indica che tutti hanno terminato chiamando il metodo *WaitFor*. *WaitFor* attende per un tempo specificato il segnale da impostare, e restituisce uno dei valori elencati nella Tabella 11.2.

Tabella 11.2 Valori di ritorno di Waifor

Valori	Significato
wrSignaled	Il segnale dell'evento è stato impostato.
wrTimeout	Il tempo specificato è trascorso senza che il segnale sia stato impostato.
wrAbandoned	L'oggetto evento è stato distrutto prima che sia trascorso il periodo di time out.
wrError	Si è verificato un errore durante l'attesa.

Il codice seguente mostra come il thread principale avvia l'esecuzione dei thread, quindi ricomincia quando tutti hanno terminato l'operazione:

```
Event1->ResetEvent(); // clear the event before launching the threads
for (int i = 0; i < Counter; i++)
  new TaskThread(false); // create and launch task threads
if (Event1->WaitFor(20000) != wrSignaled)
  throw Exception;
// now continue with the main thread, all task threads have finished
```

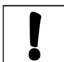


Se non si vuole fermare l'attesa di un evento dopo un determinato periodo di tempo, passare il metodo *WaitFor* con INFINITE come valore del parametro. Bisogna prestare attenzione quando si usa INFINITE, perché il thread si blocca se il segnale anticipato non viene mai ricevuto.

Esecuzione di oggetti thread

Dopo aver implementato una classe thread fornendole un metodo *Execute*, è possibile usarla nell'applicazione per lanciare il codice nel metodo *Execute*. Per usare un thread, creare innanzi tutto un'istanza della classe thread. Si può creare un'istanza del thread che avvia immediatamente l'esecuzione o il thread stesso in uno stato di sospensione, in modo che parta solo quando viene chiamato il metodo *Resume*. Per creare un thread che si avvia immediatamente, impostare il parametro *CreateSuspended* del costruttore a **false**. Per esempio, la seguente riga crea un thread e ne avvia l'esecuzione:

```
TMyThread *SecondThread = new TMyThread(false); // create and run the thread
```



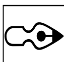
Bisogna evitare di creare troppi thread nell'applicazione. Le difficoltà di gestione di più thread possono influire negativamente sulle prestazioni. Il limite consigliato è di 16 thread per processo su sistemi monoprocessore. Questo limite presuppone che la maggior parte di quei thread sia in attesa di eventi esterni. Se i thread sono tutti attivi, se ne potranno usare di meno.

È possibile creare istanze multiple dello stesso tipo di thread per eseguire codice parallelo. Per esempio, si può lanciare una nuova istanza di un thread in risposta ad una certa operazione dell'utente, consentendo a ciascun thread l'esecuzione della risposta attesa.

Ridefinizione della priorità predefinita

Quando la quantità di tempo che la CPU deve dedicare al thread è implicita nell'operazione del thread, la sua priorità viene impostata nel costruttore. Ciò viene descritto nel paragrafo "Inizializzazione del thread" a pagina 11-3. Tuttavia, se la priorità del thread varia a seconda di quando esso viene eseguito, creare il thread in uno stato di sospensione, impostare la priorità e quindi avviare l'esecuzione del thread:

```
TMyThread *SecondThread = new TMyThread(true); // create but don't run
SecondThread->Priority = tpLower; // set the priority lower than normal
SecondThread->Resume(); // now run the thread
```



Se si scrive un'applicazione multipiattaforma, è necessario utilizzare codice differente per l'assegnazione delle priorità in Windows e Linux. In Linux, *Priority* è un valore numerico che dipende dalle politiche di threading che puossono essere cambiate solo dall'utente root. Per i dettagli, consultare la versione CLX di *TThread* e *Priority* nella Guida in linea.

Avvio e interruzione dei thread

Un thread può essere avviato ed interrotto un qualsiasi numero di volte prima che finisca l'esecuzione. Per interrompere temporaneamente un thread, chiamare il suo metodo *Suspend*. Quando non ci sono problemi nel riprendere l'esecuzione del thread, chiamare il suo metodo *Resume*. *Suspend* incrementa un contatore interno, in modo da poter annidare le chiamate a *Suspend* e *Resume*. Il thread non riprende

l'esecuzione finché tutte le sospensioni non sono state soddisfatte da una chiamata a *Resume*.

È possibile richiedere che un thread termini anzitempo l'esecuzione chiamando il metodo *Terminate*. *Terminate* imposta la proprietà *Terminated* del thread a **true**.Se il metodo *Execute* è stato implementato correttamente, controlla periodicamente la proprietà *Terminated*, e interrompe l'esecuzione quando *Terminated* è **true**.

Debug di applicazioni multi-thread

Quando si effettua il debug delle applicazioni multi-thread, il tentativo di registrare lo stato di tutti i thread che sono eseguiti simultaneamente, o anche di determinare quale thread è in esecuzione quando ci si ferma ad un breakpoint, può generare confusione. Per seguire e intervenire su tutti i thread dell'applicazione, si può usare la casella TThread Status. Per visualizzare la casella Thread Status, scegliere dal menu principale il comando View|Thread.

Quando si verifica un evento di debug (breakpoint, eccezione, pausa), la vista dello stato dei thread indica in dettaglio la situazione. Fare clic con il tasto destro del mouse sulla finestra Thread Status per accedere ai comandi che individuano la posizione corrispondente del sorgente o che rendono attivo un thread differente. Quando un thread viene contrassegnato come attivo, il passo o la successiva operazione da eseguire si riferisce a quel thread.

La finestra Thread Status elenca tutti i thread di esecuzione dell'applicazione in base al rispettivo ID. Se si usano degli oggetti thread, il loro ID è il valore della proprietà *ThreadID*. Se non si usano oggetti thread, l'ID di ciascun thread viene restituito tramite la chiamata a *CreateThread o BeginThread*.

Per ulteriori informazioni sulla finestra Thread Status, consultare la Guida in linea.

Assegnazione di un nome al thread

Dal momento che risulta difficile correlare un ID di thread con uno dei thread presenti nella finestra Thread Status, è possibile assegnare dei nomi alle classi thread. Quando si crea una classe thread nella finestra di dialogo Thread Object, oltre a immettere un nome di classe, selezionare anche la casella di controllo Named Thread, immettere un nome di thread e fare clic su OK.

L'assegnazione del nome alla classe thread aggiunge un metodo alla classe thread di nome *SetName*. Quando il thread inizia la sua esecuzione, per prima cosa chiama il metodo *SetName*.

È possibile assegnare un nome ai threads solo nelle applicazioni VCL.

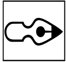
Conversione di un thread senza nome in uno con nome

È possibile convertire un thread senza nome in uno con nome. Ad esempio, se si ha una classe thread creata con C++Builder 5 o con versioni precedenti, è possibile convertirla in un thread con nome seguendo questa procedura.

1 Aggiungere alla classe thread il metodo *SetName*:

```
//---------------------------------------------------------------------------
void TMyThread::SetName()
{
        THREADNAME_INFO info;
        info.dwType = 0x1000;
        info.szName = "MyThreadName";
        info.dwThreadID = -1;
        info.dwFlags = 0;

        __try
        {
                RaiseException( 0x406D1388, 0, sizeof(info)/sizeof(DWORD),(DWORD*)&info );
        }
        __except (EXCEPTION_CONTINUE_EXECUTION)
        {
        }
}
//---------------------------------------------------------------------------
```



Impostare `info.szName` al nome della classe thread.

Il debugger rileva l'eccezione e controlla il nome del thread nella struttura che gli è stata passata. Durante le operazioni di debug, il debugger mostra il nome del thread nel campo Thread ID della finestra Thread Status.

2 Aggiungere all'inizio del metodo *Execute* una chiamata al nuovo metodo *SetName*:

```
//---------------------------------------------------------------------------
void __fastcall TMyThread::Execute()
{
        SetName();
        //---- Place existing Execute method code here ----
}
//---------------------------------------------------------------------------
```

Assegnazione di nomi differenti a thread simili

Tutte le istanze di un thread di una stessa classe thread hanno lo stesso nome. Tuttavia, in fase di esecuzione, è possibile assegnare un nome diverso a ciascuna istanza del thread seguendo questa procedura.

1 Aggiungere alla classe thread una proprietà *ThreadName* aggiungendo alla definizione della classe quanto segue:

```
__property AnsiString ThreadName = {read=FName, write=FName};
```

2 Nel metodo *SetName*, modificare la riga che riporta:

```
info.szName = “MyThreadName”;
```

in:

```
info.szName = ThreadName;
```

3 Quando si crea l'oggetto thread:

1 Crearlo in uno stato di sospensione. Consultare “Esecuzione di oggetti thread” a pagina 11-12.

2 Assegnare un nome, come ad esempio MyThread

```
MyThread.ThreadName="SearchForFiles";
```

3 Riprendere il thread. Consultare “Avvio e interruzione dei thread” a pagina 11-12.

Capitolo

12

Gestione delle eccezioni

C++Builder supporta la gestione delle eccezioni C++, la gestione delle eccezioni strutturate basata su C e la gestione delle eccezioni VCL e CLX.

È possibile sollevare qualsiasi eccezione C++ Ansi Standard oltre che eccezioni di tipo VCL, le quali includono routine native per la gestione degli errori.

Viene fornito anche il supporto per le eccezioni strutturate Win32 basate su C, in modo che il codice possa reagire correttamente alle eccezioni sollevate dal sistema operativo Windows.

Gestione delle eccezioni C++

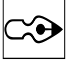
Le eccezioni sono condizioni eccezionali che richiedono una gestione speciale; possono includere errori che si verificano in fase di esecuzione, come una divisione per zero o l'esaurimento della memoria disponibile. La *gestione delle eccezioni* fornisce un modo standard per la gestione degli errori, per scoprire i problemi previsti e quelli inattesi, e per consentire agli sviluppatori di riconoscere, di tracciare e di correggere gli errori.

Quando si verifica un errore, il programma *solleva* un'eccezione. Di solito l'eccezione contiene informazioni su ciò che è accaduto. Ciò permette a un'altra parte del programma di diagnosticare la causa dell'eccezione.

I programmi vengono preparati per le eccezioni inserendo in un *blocco try* una serie di istruzioni che potrebbero provocarle. Se una di queste istruzioni solleva un'eccezione, il controllo viene trasferito a un *gestore di eccezioni* che gestisce quel tipo di eccezione. Si dice che il gestore di eccezioni *intercetta* l'eccezione e specifica le azioni da eseguire prima di terminare il programma.

Sintassi della gestione delle eccezioni

La gestione delle eccezioni richiede l'uso di tre parole chiave: **try**, **catch** e **throw**. La parola chiave throw viene utilizzata per generare un'eccezione. Il blocco try contiene istruzioni che potrebbero sollevare eccezioni ed è seguito da una o più istruzioni catch. Ogni istruzione catch gestisce un tipo specifico di eccezione.



Le parole chiave **try**, **catch** e **throw** non sono ammesse nei programmi C.

Il blocco try

Il blocco try contiene un'istruzione o una serie di istruzioni che possono sollevare un'eccezione. Un programma segnala un'eccezione eseguendo un'istruzione **throw**. L'istruzione **throw** generalmente si trova all'interno di una funzione: Per esempio:

```
void SetFieldValue(DF *dataField, int userValue)
{
    if ((userValue < 0) || userValue > 10)
      throw EIntegerRange(0, 10, userValue);
      . . .
}
```

Un'altra parte del programma può catturare l'oggetto che ha segnalato l'eccezione e gestirlo di conseguenza. Per esempio:

```
try
{
    SetFieldValue(dataField, userValue);
}
catch (EIntegerRange &rangeErr)
{
    printf("Expected value between %d and %d, but got %d\n",
         rangeErr.min, rangeErr.max, rangeErr.value);
}
```

Nell'esempio precedente, se la funzione *SetFieldValue* rileva che i suoi parametri di input non sono validi, può sollevare un'eccezione per indicare ciò. Il blocco try/catch include *SetFieldValue* per intercettare l'eccezione sollevata da *SetFieldValue* ed esegue l'istruzione printf. Se non viene sollevata alcuna eccezione, l'istruzione printf non sarà eseguita.

Un blocco try specificato da **try** deve essere seguito immediatamente dal *gestore* specificato da **catch**. Il blocco try è un'istruzione che specifica il flusso di controllo del programma. Se viene segnalata un'eccezione nel blocco try, il controllo del programma viene trasferito ad un gestore di eccezione appropriato.

Il *gestore* è un blocco di codice progettato per gestire le eccezioni. Il linguaggio C++ richiede almeno un gestore immediatamente dopo un blocco try. Il programma deve includere un gestore per ogni eccezione che può generare.

L'istruzione throw

L'istruzione throw può innescare diversi tipi di oggetti. Gli oggetti in C++ possono essere generalmente innescati per valore, per riferimento o per puntatore. Per esempio:

```
// throw an object, to be caught by value or reference
throw EIntegerRange(0, 10, userValue);

// throw an object to be caught by pointer
throw new EIntegerRange(0, 10, userValue);
```

I due esempi successivi mostrano funzionalità che vengono fornite in gran parte per la completezza nello standard. È preferibile innescare eccezioni più descrittive. Vi sono casi particolari per innescare tipi nativi, come gli integer. Inoltre, è preferibile non innescare eccezioni per puntatore.

```
// throw an integer
throw 1;

// throw a char *
throw "foo";
```

Nella maggior parte dei casi, si preferirà intercettare le eccezioni per riferimento, e in particolare mediante riferimenti di tipo **const**. Vi sono alcuni casi in cui si vorrà stare attenti ad intercettare oggetti mediante valore. Gli oggetti che vengono intercettati mediante valore devono essere copiati prima di poterli assegnare al parametro di intercettazione. Se un utente fornisce un costruttore di copia, questo costruttore verrà chiamato, e ciò può apportare inefficienza.

L'istruzione catch

L'istruzione **catch** ha diverse forme. Gli oggetti possono essere intercettati per valore, per riferimento o per puntatore. Inoltre, al parametro catch possono essere applicati i modificatori **const**. Per un singolo blocco try vi possono essere più istruzioni **catch** in modo da consentire l'intercettazione di più diversi di eccezioni da parte di un blocco, e ci dovrebbe essere un'istruzione **catch** per ogni eccezione che potrebbe essere sollevata. Per esempio:

```
try
   CommitChange(dataBase, recordMods);
catch (const EIntegerRange &rangeErr)
   printf("Got an integer range exception");
catch (const EFileError &fileErr)
   printf("Got a file I/O error");
```

Se la funzione *CommitChange* utilizza più sottosistemi e questi sottosistemi possono innescare vari tipi di eccezioni, si potrebbe voler gestire ogni tipo di eccezione in modo separato. Con più istruzioni **catch** per una singola istruzione **try**, è possibile avere gestori per ogni tipo di eccezione.

Se un oggetto eccezione è derivato da una classe base, si potrebbero volere aggiungere dei gestori specializzati per alcune eccezioni derivate, ma si potrebbe voler includere anche un gestore generico per la classe base. L'operazione è fattibile collocando delle istruzioni catch nella sequenza con cui si desidera che vengano ricercate nel momento in cui viene sollevata un'eccezione. Il codice che segue, ad esempio, gestisce dapprima *EIntegerRange* e poi *ERange*, da cui è derivato *EintegerRange*.

```
try
   SetFieldValue(dataField, userValue);
```

```
catch (const EIntegerRange &rangeErr)
   printf("Got an integer range exception");
catch (const ERange &rangeErr)
   printf("Got a range exception");
```

Infine, se si desidera che il gestore intercetti tutte le eccezioni che potrebbero essere sollevate dopo il blocco try, si utilizza la forma speciale **catch(…)**. Questa forma dice al sistema di gestione delle eccezione che si dovrebbe richiamare il gestore per qualsiasi eccezione. Per esempio:

```
try
   SetFieldValue(dataField, userValue);
catch (...)
   printf("Got an exception of some kind");
```

Risollevamento delle eccezioni

In alcuni casi, un gestore di eccezioni potrebbe elaborare un'eccezione, quindi sollevare di nuovo la stessa eccezione o sollevare un'eccezione differente.

Se il gestore vuole sollevare di nuovo l'eccezione corrente, è sufficiente che utilizzi l'istruzione **throw** senza parametri. In questo modo si dice al compilatore o alla libreria RTL di prendere l'oggetto eccezione corrente e sollevarlo di nuovo. Per esempio:

```
catch (EIntegerRange &rangeErr)
{
    // Code here to do local handling for the exception
    throw;  // rethrow the exception
}
```

Se il gestore vuole sollevare un'eccezione diversa, è sufficiente che utilizzi solo l'istruzione **throw** nel modo solito.

Specifica delle eccezioni

È possibile specificare quali sono le eccezioni che una funzione può solevare. Se si solleva un'eccezione di tipo errato nei confront di una funzione viene generato un errore di runtime. La sintassi per la *specifica delle eccezioni* è:

```
exception-specification:
   throw (type-id-list)    //type-id-list is optional
   type-id-list:
      type-id
      type-id-list, type-id
```

Gli esempi seguenti sono funzioni con specifica delle eccezioni.

```
void f1();              // The function can throw any exception

void f2() throw();      // Should not throw any exceptions

void f3() throw( A, B* ); // Can throw exceptions publicly derived from A,
                          // or a pointer to publicly derived B
```

La definizione e tutte le dichiarazioni di tale funzione devono avere una specifica delle eccezioni contenente lo stesso gruppo di *type-id*. Se una funzione segnala un'eccezione non contenuta nella sua specifica, il programma chiamerà *unexpected*.

È possibile che non si voglia specificare un'eccezione per i seguenti motivi:

Anzitutto, in Windows vi è un calo delle prestazioni runtime legato alla specifica delle eccezione in una funzione.

In secondo luogo, è possibile ottenere errori inattesi in esecuzione. Ad esempio, si supponga che il sistema utilizzi la specifica delle eccezioni e utilizzi un altro sottosistema nella sua implementazione. Ora si supponga che il sottosistema venga modificato in modo che sollevi nuovi tipi di eccezione. Quando si utilizza il nuovo codice del sottosistema, si potrebbero ottenere errori di runtime senza ottenere mai dal compilatore un'indicazione che ciò potrebbe accadere.

Infine, se si utilizzano funzioni virtuali, è possibile violare il progetto del programma. Ciò è dovuto al fatto che la specifica delle eccezioni non è considerata parte del tipo della funzione. Nell'esempio seguente la classe derivata *BETA::vfunc* è definita in modo tale da non segnalare nessuna eccezione—una deviazione rispetto alla dichiarazione originale della funzione.

```
class ALPHA {
public:
  struct ALPHA_ERR {};
  virtual void vfunc(void) throw (ALPHA_ERR) {} // Exception specification
};

class BETA : public ALPHA {
  void vfunc(void) throw() {} // Exception specification is changed
};
```

Rilascio delle eccezioni

Quando viene sollevata un'eccezione, la libreria runtime prende l'oggetto che è stato sollevato, ottiene il tipo dell'oggetto e guarda nello stack delle chiamate alla ricerca di un gestore il cui tipo corrisponda al tipo dell'oggetto gettato. Una volta trovato un gestore, la RTL rilascia lo stack fino al punto del gestore e lo esegue.

Nel processo di rilascio, la RTL chiama i distruttori per tutti gli oggetti locali presenti nei frame dello stack dal punto in cui l'eccezione è stata sollevata fino al punto in cui è stata intercettata. Se un distruttore provoca il sollevamento di un'eccezione durante il rilascio dello stack e non la gestisce, viene chiamata la funzione *terminate*. I distruttori vengono chiamati per impostazione predefinita, ma è possibile disattivare questo comportamento predefinito utilizzando l'opzione **-xd** del compilatore.

Puntatori sicuri

Nel caso si abbiano variabili locali che sono puntatori a oggetti e viene sollevata un'eccezione, questi puntatori non vengono cancellati automaticamente. Ciò accade perché il compilatore poiché non ha alcun modo per distinguere fra un puntatore ai dati, allocato solo per questa funzione, e un qualsiasi altro puntatore. La classe che è possibile utilizzare, per essere certi che gli oggetti allocati per un utilizzo locale siano

distrutti nel caso in cui si verifichi un'eccezione, è auto_ptr. C'è un caso particolare in cui la memoria per un puntatore allocato in una funzione viene liberata:

```
TMyObject *pMyObject = new TMyObject;
```

In questo esempio, se il costruttore per *TMyObject* solleva un'eccezione, il puntatore all'oggetto allocato per *TMYObject* sarà cancellato dalla RTL nel momento in cui rilascia l'eccezione. Questo è l'unico caso in cui il compilatore cancella automaticamente un valore di puntatore.

I costruttori nella gestione delle eccezioni

I costruttori delle classi possono segnalare eccezioni se non sono in grado di costruire un oggetto. Se un costruttore segnala un'eccezione, non viene chiamato necessariamente il distruttore dell'oggetto. I distruttori vengono chiamati solo per le classi base e per gli oggetti che sono stati completamente costruiti all'interno delle classi a partire dall'ingresso nel blocco **try**.

Gestione di eccezioni non intercettate e inattese

Se viene sollavata un'eccezione e non viene trovato nessun gestore dell'eccezione—cioè, l'eccezione non viene intercettata—il programma chiama una funzione di terminazione. È possibile specificare una propria funzione di terminazione con *set_terminate*. Se non si specifica una funzione di terminazione, viene chiamata la funzione *terminate*. Per esempio, il codice seguente utilizza la funzione *my_terminate* per gestire le eccezioni non intercettate da alcun gestore.

```
void SetFieldValue(DF *dataField, int userValue)
{
  if ((userValue < 0) || (userValue) > 10));
    throw EIntegerRange(0, 10, userValue);
  . . .
}

void my_terminate()
{
  printf("Exception not caught");
  abort();
}

// Set my_terminate() as the termination function
set_terminate(my_terminate);
// Call SetFieldValue. This generates an exception because the user value is greater
// than 10. Because the call is not in a try block, my_terminate is called.
SetFieldValue(DF, 11);
```

Se una funzione specifica quali sono le eccezioni sollevate e solleva un'eccezione non specificata, viene chiamata una funzione unexpected. È possibile specificare la funzione unexpected con *set_unexpected*. Se non si specifica una funzione unexpected, viene chiamata la funzione *unexpected*.

Eccezioni strutturate in ambiente Win32

Win32 supporta la gestione delle eccezioni strutturate basate su C, che sono simili alle eccezioni di C++. Esistono, comunque, alcune differenze fondamentali che richiedono un utilizzo attento quando vengono mescolate con codice C++ per la gestione delle eccezioni.

Quando si utilizza la gestione delle eccezioni strutturate nelle applicazioni C++Builder, occorre tenere presente quanto segue:

- Le eccezioni strutturate del C possono essere utilizzate in programmi C++.
- Le eccezioni C++ non possono essere utilizzate in programmi C, in quanto richiedono che il loro gestore sia specificato tramite la parola chiave **catch**, e **catch** non è consentita nei programmi C.
- Un'eccezione generata da una chiamata alla funzione *RaiseException* viene gestita mediante il blocco **try/__except** (C++) o **__try/__except** (C). (Si possono anche utilizzare i blocchi **try/__finally** oppure **__try/__finally**. Vedere "Sintassi delle eccezioni strutturate" a pagina 12-7.) Tutti i gestori dei blocchi **try/catch** vengono ignorati quando viene chiamata la funzione *RaiseException*.
- Le eccezioni che non sono gestite dalle applicazioni non generano una chiamata a *terminate()*, ma vengono passate invece al sistema operativo (in generale, il risultato finale è la fine del processo).
- I gestori di eccezioni non ricevono una copia dell'oggetto eccezione, a meno che non la richiedano.

Nei programmi C e C++ si possono utilizzare le seguenti funzioni di supporto delle eccezioni C:

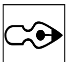
- *GetExceptionCode*
- *GetExceptionInformation*
- *SetUnhandledExceptionFilter*
- *UnhandledExceptionFilter*

C++Builder non limita l'utilizzo della funzione UnhandledExceptionFilter al filtro di eccezione dei blocchi **__try/__except** o **try/__except**. Comunque, il comportamento del programma non è definito quando questa funzione viene chiamata al di fuori del blocco **__try/__except** o **try/__except**.

Le applicazioni CLX in esecuzione su Linux non supportano eccezioni strutturate C/C++.

Sintassi delle eccezioni strutturate

In un programma C le parole chiave ANSI compatibili utilizzate per implementare le eccezioni strutturate sono **__except**, **__finally** e **__try**.



La parola chiave **__try** può apparire soltanto in programmi C. Se si vuole scrivere del codice portabile, nei programmi C++ non si deve utilizzare la gestione delle eccezioni strutturate.

Sintassi per la gestione delle eccezioni try-except

La sintassi per la gestione delle eccezioni *try-except* è la seguente:

```
blocco try:
    istruzione composta __try (in un modulo C)
    istruzione composta try (in un modulo a C++)
gestore:
    __except (espressione) istruzione composta
```

Sintassi della terminazione try-finally

Nel caso della terminazione *try-finally* la sintassi è::

```
blocco try:
    istruzione composta __try (in un modulo C)
    istruzione composta try (in un modulo a C++)
terminazione:
    __finally istruzione composta
```

Gestione delle eccezioni strutturate

Le eccezioni strutturate possono essere gestite utilizzando un'estensione della gestione delle eccezioni di C++:

```
try {
        foo();
    }
    __except(__expr__) {
        // handler here
    }
```

`__expr__` è un'espressione che può assumere uno dei tre valori seguenti:

Valore	Descrizione
EXCEPTION_CONTINUE_SEARCH (0)	Il gestore delle eccezioni non è stato indicato e il sistema operativo continua a cercarlo.
EXCEPTION_CONTINUE_EXECUTION (-1)	Continua l'esecuzione nel punto dell'eccezione.
EXCEPTION_EXECUTE_HANDLER (1)	Attiva il gestore delle eccezioni. Se il codice è stato compilato con il distruttore di cleanup abilitato (-xd, opzione attiva per impostazione predefinita), tutti i distruttori per gli oggetti locali creati tra il punto di eccezione e il gestore di eccezioni vengono chiamati quando viene scaricato lo stack. Lo scarico dello stack viene completato prima dell'attivazione del gestore.

Win32 fornisce due funzioni che possono essere utilizzate per ricercare informazioni sull'eccezione attiva: *GetExceptionCode()* e *GetExceptionInformation()*. Se si vuole chiamare una funzione come parte dell'espressione "filtro" sopra indicata, queste funzioni devono essere chiamate direttamente dall'interno del contesto di *__except()*:

```
#include <Windows.h>
#include <excpt.h>
```

```
int filter_func(EXCEPTION_POINTERS *);
...
EXCEPTION_POINTERS *xp = 0;
    try {
        foo();
    }
    __except(filter_func(xp = GetExceptionInformation())) {
        //...
    }
```

Oppure, se si preferisce utilizzare l'operatore virgola per assegnazioni annidate nelle chiamate a funzioni, vedere l'esempio seguente:

```
__except((xp = GetExceptionInformation()), filter_func(xp))
```

Filtri delle eccezioni

Un'espressione filtro può richiamare una funzione filtro, ma la funzione filtro non può chiamare *GetExceptionInformation*. È possibile passare alla funzione filtro il valore restituito da *GetExceptionInformation* come parametro.

Per passare l'informazione EXCEPTION_POINTERS ad un gestore di eccezioni, l'espressione o la funzione filtro devono copiare il puntatore o i dati da *GetExceptionInformation* in una posizione a cui il gestore può accedere successivamente.

Nel caso di istruzioni **try-except** annidate, ogni espressione filtro dell'istruzione viene valutata finché non viene individuato EXCEPTION_EXECUTE_HANDLER o EXCEPTION_CONTINUE_EXECUTION. Un'espressione filtro può richiamare *GetExceptionInformation* per ottenere informazioni sull'eccezione.

A patto che *GetExceptionInformation* o *GetExceptionCode* siano chiamate direttamente nell'espressione fornita a **__except**, si può usare una funzione per determinare cosa fare con un'eccezione anziché tentare di creare un'espressione C++ complessa. Quasi tutte le informazioni necessarie per gestire un'eccezione possono essere ricavate dal risultato di *GetExceptionInformation()*. *GetExceptionInformation()* restituisce un puntatore ad una struttura EXCEPTION_POINTERS:

```
struct EXCEPTION_POINTERS {
        EXCEPTION_RECORD *ExceptionRecord;
        CONTEXT *Context;
    };
```

EXCEPTION_RECORD contiene la stato indipendente dalla macchina:

```
struct EXCEPTION_RECORD {
        DWORD ExceptionCode;
        DWORD ExceptionFlags;
        struct EXCEPTION_RECORD *ExceptionRecord;
        void *ExceptionAddress;
        DWORD NumberParameters;
        DWORD ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
    };
```

Generalmente, la funzione filtro analizza le informazioni contenute in ExceptionRecord per decidere come rispondere. Talvolta si ha bisogno di informazioni più specifiche (specialmente se l'azione da intrapendere è EXCEPTION_CONTINUE_EXECUTION: se non si fa nulla, il codice che ha causato l'eccezione sarà eseguito nuovamente). In questi casi, l'altro campo della struttura EXCEPTION_POINTERS fornisce lo stato del processore al momento dell'eccezione. Se questa struttura viene modificata e il filtro restituisce EXCEPTION_CONTINUE_EXCEPTION, tale valore viene usato per impostare lo stato del thread prima di procedere con l'esecuzione. Per esempio:

```
static int xfilter(EXCEPTION_POINTERS *xp)
    {
        int rc;

        EXCEPTION_RECORD *xr = xp->ExceptionRecord;
        CONTEXT *xc = xp->Context;

        switch (xr->ExceptionCode) {
            case EXCEPTION_BREAKPOINT:
                // whoops, someone left an embedded breakpoint.
                // just step over it (1 byte on x86)
                ++xc->Eip;
                rc = EXCEPTION_CONTINUE_EXECUTION;
                break;

            case EXCEPTION_ACCESS_VIOLATION:
                rc = EXCEPTION_EXECUTE_HANDLER;
                break;

            default:
                // give up
                rc = EXCEPTION_CONTINUE_SEARCH;
                break;
        };

        return rc;
    }
    ...

    EXCEPTION_POINTERS *xp;

    try {
        func();
    }
    __except(xfilter(xp = GetExceptionInformation())) {
        abort();
    }
```

Abbinamento di eccezioni strutturate con C++

Bisogna essere consapevoli delle conseguenze derivanti dall'utilizzo delle eccezioni strutturate nei programmi C++. Innanzi tutto, anche se C++Builder implementa le

eccezioni C++ con le eccezioni strutturate Win32, le eccezioni C++ sono trasparenti ad un blocco **__except**.

Un blocco **try** può essere seguito esattamente da un blocco **except** o da almeno un blocco catch. Il tentativo di abbinarli genera un errore in fase di compilazione. Il codice necessario per gestire entrambi i tipi di eccezione deve essere semplicemente annidato all'interno di due blocchi **try**:

```
try {
        EXCEPTION_POINTERS *xp;

        try {
            func();
        }
        __except(xfilter(xp = GetExceptionInformation())) {
            //...
        }
    }
    catch (...) {
        //...
    }
```

La specifica di una funzione *throw()* non incide sul comportamento di un programma rispetto ad un'eccezione Win32. Inoltre, un'eccezione non gestita viene eventualmente gestita dal sistema operativo (se un debugger non la gestisce prima), a differenza di quanto fanno i programmi C++ che chiamano *terminate()*.

Tutti i moduli compilati con l'opzione **-xd** del compilatore (attiva per impostazione predefinita) richiameranno i distruttori di tutti gli oggetti con memorizzazione di tipo **auto**. Si ha lo scarico dello stack dal punto in cui si è verificata l'eccezione fino al punto in cui essa viene catturata.

Eccezioni basate su C in un programma di esempio C++

```
/* Program results:
Another exception:
Caught a C-based exception.
Caught C++ exception[Hardware error: Divide by 0]
C++ allows __finally too!
*/
#include <stdio.h>
#include <string.h>
#include <windows.h>

class Exception
{
  public:
  Exception(char* s = "Unknown"){ what = strdup(s);      }
  Exception(const Exception& e ){ what = strdup(e.what); }
 ~Exception()                   { delete[] what;         }
  char* msg() const             { return what;           }
private:
  char* what;
};
```

```
int main()
{
  float e, f, g;
  try
  {
    try
    {
      f = 1.0;
      g = 0.0;
      try
      {
        puts("Another exception:");
        e = f / g;
      }
      __except(EXCEPTION_EXECUTE_HANDLER)
      {
        puts("Caught a C-based exception.");
        throw(Exception("Hardware error: Divide by 0"));
      }
    }
    catch(const Exception& e)
    {
      printf("Caught C++ Exception: %s :\n", e.msg());
    }
  }
  __finally
  {
    puts("C++ allows __finally too!");
  }
return e;
}
```

Definizione delle eccezioni

Sollevare un'eccezione Win32 che viene gestita all'interno dello stesso programma generalmente non ha molto senso: le eccezioni C++ possono fare di meglio, risultano molto più portabili e utilizzano una sintassi più semplice. Le eccezioni Win32 offrono comunque il vantaggio che possono essere gestite da componenti che potrebbero non essere stati compilati con lo stesso compilatore C++.

Il primo passo consiste nel definire l'eccezione. Un' eccezione è un intero a 32 bit con il seguente formato (a partire dal bit 0):

Bit	Significato
31-30	11 = Errore (normale)
	00 = Esito positivo,
	01 = Informazione
	10 = Avvertenza
29	1 = Definito dall'utente

Bit	Significato
28	Riservato
27-0	Definito dall'utente

Oltre alla definizione del codice dell'eccezione, occorre decidere se includere o meno informazioni addizionali sull'eccezione (accessibile al filtro/gestore dal record di eccezione). Non ci sono metodi convenzionali per codificare altri parametri nel codice dell'eccezione. Per maggiori informazioni, consultare la documentazione su Win32 (disponibile nella Guida in linea di C++Builder).

Sollevamento delle eccezioni

Un'eccezione Win32 viene sollevata mediante una chiamata a *RaiseException()*, la quale viene dichiarata come segue

```
void RaiseException(DWORD ec, DWORD ef, DWORD na, const DWORD *a);
```

dove:

ec Codice di eccezione

ef Exception flags, either 0 or EXCEPTION_NONCONTINUABLE

(Flag di eccezione; assume il valore 0 o EXCEPTION_NONCONTINUABLE. (Se l'eccezione viene contrassegnata come non continuabile e un filtro prova a continuarla, viene sollevata EXCEPTION_NONCONTINUABLE_EXCEPTION.)

na Numero di elementi dell'array di argomenti

a Puntatore al primo elemento dell'array di argomenti–il significato di questi argomenti dipende dalla specifica eccezione

Blocchi di terminazione

Il modello di gestione delle eccezioni strutturate supporta un "blocco di terminazione", che viene eseguito sia che si esca normalmente da un blocco controllato sia che si esca mediante un'eccezione. Il compilatore di C++Builder lo supporta in C con la seguente sintassi:

```
__try {
        func();
    }
    __finally {
        // this happens whether func() raises an exception or not
    }
```

I blocchi di terminazione sono supportati da un'estensione C++ dove è possibile gestire le operazioni di pulizia nel blocco **__finally**:

```
try {
        func();
    }
    __finally {
```

```
        // this happens whether func() raises an exception or not
    }
```

L’esempio seguente illustra i blocchi di terminazione:

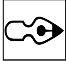
```
/* Program results:
An exception:
Caught an exception.
The __finally is executed too!
No exception:
No exception happened, but __finally still executes!
*/
#include <stdio.h>
#include <windows.h>

int main()
{
  float e, f, g;

  try
  {
    f = 1.0;
    g = 0.0;
    try
    {
      puts("An exception:");
      e = f / g;
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
    {
      puts("Caught an exception.");
    }
  }
  __finally
  {
    puts("The __finally is executed too!");
  }
  try
  {
    f = 1.0;
    g = 2.0;
    try
    {
      puts("No exception:");
      e = f / g;
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
    {
      puts("Caught an exception.");
    }
  }
  __finally
  {
    puts("No exception happened, but __finally still executes!");
  }
```

```
return e;
}
```

Il codice C++ può anche gestire un "blocco di terminazione" creando oggetti locali con i distruttori che vengono chiamati quando si esce dal campo d'azione. Dato che le eccezioni strutturate di C++Builder supportano le operazioni di pulizia del distruttore, questo funzionerà indipendentemente dal tipo di eccezione che si è verificata.



Si ha una situazione particolare nel caso in cui si verifica un'eccezione non gestita dal programma. Per un'eccezione C++ il compilatore di C++Builder chiama i distruttori degli oggetti locali (non richiesto dalla definizione del linguaggio), mentre con un'eccezione Win32 non gestita l'operazione di pulzia del distruttore non avviene.

Opzioni di C++Builder per la gestione delle eccezioni

Di seguito sono riportate le opzioni di gestione delle eccezioni del compilatore di C++Builder.

Tabella 12.1 Opzioni del compilatore per la gestione delle eccezioni

Opzione della riga comandi	Descrizione
`-x`	Abilita la gestione delle eccezioni di C++ (Attiva per impostazione predefinita))
`-xd`	Abilita la cancellazione da parte del distruttore. Chiama i distruttori di tutti gli oggetti dichiarati in modo automatico all'interno del campo d'azione delle istruzioni catch e throw quando viene segnalata un'eccezione. (Opzione evoluta-attiva per impostazione predefinita)
`-xp`	Abilita le informazioni sulla posizione dell'eccezione. Rende disponibile in fase di esecuzione l'identificazione delle eccezioni fornendo i numeri di riga nel codice sorgente in cui è si trova l'eccezione. Ciò consente al programma di individuare il file e il numero di riga in cui si è verificata un'eccezione C++ usando le definizioni globali __ThrowFileName e __ThrowLineNumber. (Opzione evoluta)

Gestione delle eccezioni VCL/CLX

Quando nelle proprie applicazioni si utilizzano componenti VCL/CLX, occorre comprendere il relativo meccanismo per la gestione delle eccezioni. Le eccezioni sono, infatti, integrate in molte classi e vengono segnalate automaticamente quando avviene qualcosa di inatteso. Se non si gestisce l'eccezione, la VCL e la CLX la gestirà in maniera predefinita. Generalmente viene visualizzato un messaggio che descrive il tipo di errore verificatosi.

Quando in fase di stesura del programma si incontra un'eccezione che visualizza un messaggio che indica il tipo di eccezione che è stata segnalata, è possibile controllare la classe dell'eccezione nella Guida in linea. Le informazioni fornite spesso permettono di determinare dove si è verificato l'errore e quale ne è la causa.

Inoltre, nel Capitolo 13, “Supporto del linguaggio C++ per la VCL e la CLX” vengono descritte le sottili differenze di linguaggio che possono provocare eccezioni. La sezione “Eccezioni sollevate dai costruttori” a pagina 13-14 riporta un esempio che mostra cosa accade se viene sollevata un’eccezione durante la costruzione di un oggetto.

Differenze tra C++ e VCL/CLX nella gestione delle eccezioni

Di seguito sono riportate alcune differenze molto importanti tra notevoli tra C++ e VCL/CLX nella gestione delle eccezioni.

Eccezioni sollevate dai costruttori:

- I distruttori C++ vengono chiamati per i membri e le classi basi che sono stati completamente costruiti.
- I distruttori della classe base VCL vengono chiamati anche se l’oggetto o la classe base non sono stati completamente costruiti.

Intercettazione e sollevamento di eccezioni:

- Le eccezioni C++ possono essere intercettate per riferimento, per puntatore o per valore. Le eccezioni VCL o CLX, che sono eccezioni derivate da *TObject*, possono essere catturate solo per riferimento o per puntatore. Il tentativo di interettare le eccezioni di *TObject* per valore restituisce un errore in fase di compilazione. Le eccezioni dell’hardware o del sistema operativo, come *EAccessViolation*, devono essere intercettate per riferimento.
- Le eccezioni VCL e CLX vengono intercettate per riferimento.
- Non si può utilizzare **throw** per sollevare nuovamente un’eccezione che è stata intercettata all’interno del codice VCL o CLX.

Gestione delle eccezioni del sistema operativo

C++Builder permette di gestire le eccezioni segnalate dal sistema operativo. Queste includono le violazioni di accesso, gli errori matematici sugli interi, gli errori matematici in virgola mobile, l’overflow dello stack e gli interrupt di *Ctrl+C*. Tali eccezioni vengono gestite in RTL C++ e convertite in oggetti di classe eccezione della VCL prima di essere inviate all’applicazione. Si può scrivere del codice come questo:

```
try
{
  char * p = 0;
  *p = 0;
}
// You should always catch by reference.
catch (const EAccessViolation &e)
{
  printf("You can't do that!\n");
}
```

Le classi utilizzate da C++Builder sono le stesse utilizzate da Delphi e sono disponibili soltanto per le applicazioni VCL e CLX di C++Builder. Sono derivate da *TObject* e richiedono di appoggiarsi alla VCL o alla CLX.

Di seguito sono riportate alcune caratteristiche della gestione delle eccezioni di C++Builder:

- **Non** si è responsabili del rilascio dell'oggetto eccezione.
- Le eccezioni del sistema operativo devono essere intercettate per riferimento.
- Non si può sollevare di nuovo un'eccezione di sistema operativo una volta che si è usciti dal frame di intercettazione e l'operazione di intercettazione è stata effettuata tramite i frame di intercettazione della VCL o della CLX.
- Non si può sollevare di nuovo un'eccezione di sistema operativo una volta che si è usciti dal frame di intercettazione e l'operazione di intercettazione è stata effettuata tramite i frame di intercettazione del sistema operativo.

Gli ultimi due punti possono essere sintetizzati in questo modo: una volta che l'eccezione di sistema operativo è stata intercettata come un'eccezione C++, essa non può essere sollevata di nuovo come un+eccezione di sistema operativo o come un+eccezione della VCL o della CLX, a meno che non ci si trovi nella struttura dello stack di intercettazione.

Gestione delle eccezioni della VCL e della CLX

C++Builder amplia la semantica per la gestione delle eccezioni software sollevate dalla VCL e dalla CLX o, analogamente, delle eccezioni sollevate da C++, in cui la classe di eccezione da sollevare è derivata da *TObject*. In questo caso, si applica una coppia di regole derivanti dal fatto che le classi in stile VCL possono essere allocate solo nello heap.

- Le classi di eccezione in stile VCL possono essere intercettate solo per puntatore, se si tratta di un'eccezione software, o per riferimento (che è la modalità preferita).
- Un'eccezione in stile VCL deve essere sollevata mediante la sintassi "per valore".

Classi di eccezioni VCL e CLX

C++Builder include un vasto gruppo di classi di eccezione native per la gestione automatica degli errori di divisione per zero, degli errori di I/O su file, degli errori di conversione di tipo e di molte altre condizioni di eccezione. Tutte le classi di eccezione VCL e CLX discendono da un oggetto radice di nome *Exception*. *Exception* incapsula le proprietà fondamentali e i metodi per tutte le eccezioni ti tipo VCL e fornisce un'interfaccia compatibile in modo che le applicazioni gestiscano le eccezioni.

Le eccezioni possono essere passate ad un blocco **catch** che accetta un parametro di tipo *Exception*. Per intercettare le eccezioni VCL e CLX, utilizzare la seguente sintassi:

```
catch (exception_class &exception_variable)
```

Si deve specificare la classe di eccezione che si vuole intercettare e fornire una variabile tramite la quale fare riferimento all'eccezione.

L'esempio seguente mostra come sollevare un'eccezione VCL o CLX:

```
void __fastcall TForm1::ThrowException(TObject *Sender)
{
  try
  {
    throw Exception("An error has occurred");
  }
  catch(const Exception &E)
  {
    ShowMessage(AnsiString(E.ClassName())+ E.Message);
  }
}
```

L'istruzione **throw** nell'esempio precedente crea un'istanza della classe *Exception* e chiama il suo costruttore. Tutte le eccezioni che discendono da *Exception* hanno un messaggio che può essere visualizzato, passato attraverso i costruttori, e recuperato tramite la proprietà *Message*.

Le classi di eccezione VCL/CLX selezionate sono descritte nella Tabella 12.2.

Tabella 12.2 Classi di eccezione selezionate

Classe di eccezione	Descrizione
EAbort	Interrompe una sequenza di eventi senza visualizzare una finestra di dialogo con un messaggio di errore.
EAccessViolation	Controlla gli errori di accesso non valido alla memoria.
EBitsError	Impedisce tentativi di accesso non valido agli array booleani.
EComponentError	Segnala un tentativo di registrare o di rinominare un componente.
EConvertError	Indica errori di conversione di oggetti o di stringhe.
EDatabaseError	Specifica un errore di accesso al database.
EDBEditError	Cattura i dati incompatibili con una maschera specificata.
EDivByZero	Cattura gli errori di divisione per zero.
EExternalException	Indica un codice di eccezione non riconosciuto.
EInOutError	Rappresenta un errore di I/O su file.
EIntOverflow	Specifica calcoli interi i cui risultati sono troppo grandi per il registro allocato register.
EInvalidCast	Controlla gli errori di conversione di tipo non consentito.
EInvalidGraphic	Indica un tentativo di operare con un formato grafico non riconosciuto.
EInvalidOperation	Si verifica quando si tentano operazioni non consentite su un componente.
EInvalidPointer	Risultato di operazioni di puntatore non valide.
EMenuError	È relativo a un problema con una voce di menu.
EOleCtrlError	Rivela problemi con i collegamenti ai controlli ActiveX.
EOleError	Specifica errori di automazione OLE.
EPrinterError	Segnala un errore di stampa.

Tabella 12.2 Classi di eccezione selezionate

Classe di eccezione	Descrizione
EPropertyError	Si verifica per un tentativo non riuscito di impostare il valore di una proprietà.
ERangeError	Indica un valore intero troppo grande per il tipo dichiarato a cui è stato assegnato.
ERegistryException	Specifica errori di registro.
EZeroDivide	Cattura gli errori di divisione per zero in virgola mobile.

Come si può osservare dall'elenco sopra riportato, le classi di eccezione VCL e CLX native gestiscono la maggior parte delle eccezioni, semplificando così il codice. Vi sono altri casi in cui sarà necessario creare delle proprie classi per gestire situazioni particolari. È possibile dichiarare una nuova classe di eccezioni facendola discendere dal tipo *Exception* e creando tutti i costruttori necessari (o copiandoli da una classe esistente in Sysutils.hpp).

Considerazioni sulla portabilità

Con C++Builder vengono fornite diverse librerie runtime (RTL). La maggior parte di esse riguardano le applicazioni C++Builder, ma una (cw32mt.lib) è la normale RTL multi-thread che non fa alcun riferimento alla VCL o alla CLX. Questa RTL viene fornita per il supporto di applicazioni preesistenti che devono essere inserite in un progetto, ma che non devono dipendere dalla VCL. Questa RTL non è in grado di gestire le eccezioni del sistema operativo, perché si tratta di oggetti eccezione derivati da *TObject* e richiederebbe che l'applicazione venisse collegata con parti della VCL o della CLX.

È possibile utilizzare la libreria cp32mt.lib una libreria runtime multi-thread che fornisce la gestione della memoria e la gestione delle eccezioni con la VCL e la CLX.

È possibile utilizzare due librerie di importazione, cw32mti.lib e cp32mti.lib, per l'utilizzo della DLL della RTL. Utilizzare cp32mti.lib per il supporto delle eccezioni VCL e CLX.

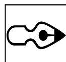
Capitolo

13

Supporto del linguaggio C++ per la VCL e la CLX

C++Builder sfrutta le funzionalità Rapid Application Development (RAD) della Visual Component Library (VCL) e della Component Library for Cross-Platform (CLX) scritta in Object Pascal. Questo capitolo spiega come le funzionalità, i costrutti e i concetti del linguaggio Object Pascal sono stati implementati in C++Builder in modo da supportare la VCL e la CLX. Queste pagine sono rivolte ai programmatori che usano oggetti VCL e CLX nelle loro applicazioni e agli sviluppatori che creano nuove classi derivate da classi VCL e CLX.

Nella prima parte di questo capitolo vengono confrontati i modelli degli oggetti C++ e Object Pascal, e viene descritto il modo in cui C++Builder abbina questi due approcci. Nella seconda parte del capitolo viene illustrato il modo in cui i costrutti del linguaggio Object Pascal vengono tradotti in C++ e in C++Builder. Vi sono riportate informazioni dettagliate sulle estensioni delle parole chiave che sono state aggiunte per supportare la VCL e la CLX. Alcune di queste estensioni, come closure e proprietà, sono utili funzionalità, indipendenti dal supporto al codice basato su VCL.



I riferimenti alle classi C++ derivate da *TObject* riguardano le classi per le quali *TObject* è il migliore antenato, ma non necessariamente il genitore da cui discendono direttamente. Per coerenza con la diagnostica del compilatore a queste classi viene anche fatto riferimento come a “classi in stile VCL”.

Modelli di oggetti C++ e Object Pascal

I modelli delle classi C++ e Object Pascal presentano diversità sia molto evidenti sia sottili. Una delle differenze più evidenti è che C++ consente l’eredità multipla mentre Object Pascal è limitato a un modello di eredità singola. Inoltre, C++ e Object Pascal presentano delle piccole differenze nel modo in cui creano, inizializzano, fanno riferimento, copiano e distruggono gli oggetti. In questa sezione vengono descritte queste differenze e il loro impatto sulle classi stile VCL di C++Builder.

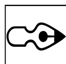
Eredità e interfacce

A differenza di C++, il linguaggio Object Pascal non supporta l'eredità multipla. Qualsiasi classe creata che abbia come antenati della VCL o della CLX eredita questa limitazione. Ciò significa che è impossibile utilizzare più classi di base per una classe C++ in stile VCL, anche nel caso in cui la classe VCL o CLX non sia l'antenato immediato.

Utilizzo delle interfacce invece dell'eredità multipla

Per molte situazioni in cui in C++ si utilizzerebbe l'eredità multipla, il codice Object Pascal utilizza invece le interfacce. Non esiste alcun costrutto in C++ che corrisponde direttamente al concetto di interfaccia dell' Object Pascal. Un'interfaccia Object Pascal agisce come una classe senza implementazione. Cioè significa che un'interfaccia è simile a una classe in cui tutti i metodi sono pure virtual e non esiste alcun membro dati. Mentre una classe Object Pascal può avere solo una singola classe genitore, può supportare un numero qualsiasi di interfacce. Il codice Object Pascal può assegnare un'istanza di classe a variabili di uno qualunque di quei tipi di interfaccia, esattamente come può assegnare l'istanza di classe a una variabile di qualsiasi tipo di classe antenato. Ciò consente il comportamento polimorfo per quelle classi che condividono la stessa interfaccia, anche se esse non hanno un antenato comune.

In C++Builder, il compilatore riconosce le classi che hanno solo metodi pure virtual e non hanno alcun membro dati come corrispondenti a interfacce di Object Pascal. Pertanto, quando si crea una classe in stile VCL, è consentito utilizzare l'eredità multipla, ma solo se tutte le classi di base, tranne quella che è una classe della VCL, della CLX o in stile VCL, non hanno alcun membro dati e hanno solo metodi pure virtual.



Non è necessario che le classi dell'interfaccia siano classi in style VCL, l'unico requisito è che esse non abbiano alcun membro dati e abbiano solo metodi pure virtuale.

Dichiarazione delle classi interfaccia

È possibile dichiarare una classe che rappresenta un'interfaccia esattamente come qualsiasi altra classe C++. Tuttavia, utilizzando alcune convenzioni, è possibile rendere più chiaro che la classe è progettata per agire come un'interfaccia. Queste convenzioni sono le seguenti:

- Invece di utilizzare la parola chiave class, le interfacce vengono dichiarate utilizzando __interface. __interface è una macro che corrispondere con la parola chiave class. Non è necessario, ma rende più chiaro che la classe è progettata per agire come un'interfaccia.
- Di solito le interfacce hanno nomi che iniziano con la lettera 'I'. Esempi di interfaccia sono *IComponentEditor* o *IDesigner*. Se si segue questa convenzione, non è necessario controllare la dichiarazione della classe per percepire che la classe sta agendo come un'interfaccia.
- Di solito le interfacce hanno un GUID associato. Questo non è un requisito assoluto, ma la maggior parte del codice che supporta le interfacce si aspetta di

trovare un GUID. + possibile utilizzare il modificatore **__declspec** con l'argomento **uuid** per associare un'interfaccia con un GUID. Nel caso delle interfacce, la macro INTERFACE_UUID mappa alla stessa cosa.

La seguente dichiarazione di interfaccia illustra queste convenzioni:

```
__interface INTERFACE_UUID("{C527B88F-3F8E-1134-80e0-01A04F57B270}") IHelloWorld :
    public IInterface
{
public:
  virtual void __stdcall  SayHelloWorld(void) = 0 ;
};
```

Di solito, quando si dichiara una classe interfaccia, il codice C++Builder dichiara anche una classe *DelphiInterface* corrispondente che semplifica l'operatività con l'interfaccia:

```
typedef System::DelphiInterface< IHelloWorld >  _di_IHelloWorld;
```

Per informazioni sulla classe *DelphiInterface*, vedere "Interfacce Delphi" a pagina 13-21.

IUnknown e IInterface

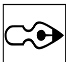
Tutte le interfacce di Object Pascal discendono da un singolo antenato comune, *IInterface*. Non è necessario che le classi interfaccia di C++ utilizzino *IInterface* come classe di base, nel senso che una classe in stile VCL può utilizzarle come classi di base aggiuntive, ma il codice VCL e CLX che opera con le interfacce presume che siano presenti i metodi di *IInterface*.

Nella programmazione COM, tutte le interfacce discendono da *IUnknown*. Il supporto COM nella VCL è basato su una definizione di *IUnknown* che mappa direttamente a *IInterface*. Ciò significa che, in Object Pascal, *IUnknown* e *IInterface* sono identici.

La definizione Object Pascal di *IUnknown*, tuttavia, non corrisponde alla definizione di *IUnknown* utilizzata in C++Builder. Il file unknwn.h definisce *IUnknown* in modo tale da includere i seguenti tre metodi:

```
virtual HRESULT STDCALL QueryInterface( const GUID &guid, void ** ppv ) = 0;
virtual ULONG STDCALL AddRef() = 0;
virtual ULONG STDCALL Release() = 0;
```

Ciò corrisponde alla definizione di *IUnknown* che è definito da Microsoft come parte delle specifiche COM.



Per informazioni su *IUnknown* e sul suo utilizzo, vedere il Capitolo 38, "Panoramica sulle tecnologie COM" oppure consultare la documentazione Microsoft.

A differenza di quanto accade in Object Pascal, la definizione di *IInterface* di C++Builder non è equivalente alla definizione di *IUnknown*. *IInterface* è invece un discendente di *IUnknown* che comprende un metodo aggiuntivo, *Supports*:

```
template <typename T>
HRESULT __stdcall  Supports(DelphiInterface<T>& smartIntf)
{
  return QueryInterface(__uuidof(T),
```

```
        reinterpret_cast<void**>(static_cast<T**>(&smartIntf)));
}
```

Supports consente di ottenere dall'oggetto che implementa *IInterface* una *DelphiInterface* per un'altra interfaccia supportata. Così, ad esempio, se si ha una *DelphiInterface* per un'interfaccia *IMyFirstInterface* e l'oggetto che implementa implementa anche *IMySecondInterface* (il cui tipo *DelphiInterface* è *_di_IMySecondInterface*), è possibile ottenere la *DelphiInterface* per la seconda interfaccia nel seguente modo:

```
_di_IMySecondInterface MySecondIntf;
MyFirstIntf->Supports(MySecondIntf);
```

La VCL e la CLX utilizzano una mappatura di *IUnknown* in Object Pascal. Questa mappatura converte i tipi utilizzati nei metodi *IUnknown* in tipi Object Pascal e rinomina i metodi *AddRef* e *Release* in *_AddRef* e in *_Release* per indicare che essi non dovrebbero essere mai chiamati direttamente. (in Object Pascal, il compilatore genera automaticamente le chiamate necessarie a metodi *IUnknown*). Quando si mappano nuovamente in C++ i metodi di *IUnknown* (o di *IInterface*) supportati dagli oggetti della VCL o della CLX, ciò produce le seguenti firme di metodi:

```
virtual HRESULT __stdcall QueryInterface(const GUID &IID, void *Obj);
int __stdcall _AddRef(void);
int __stdcall _Release(void);
```

Ciò significa che gli oggetti VCL e CLX che supportano *IUnknown* o *IInterface* in Object Pascal non supportano le versioni di *IUnknown* e *IInterface* presenti in C++Builder.

Creazione di classi che supportano IUnknown

Molte classi nella VCL e nella CLX prevedono il supporto delle interfacce. Infatti, la classe di base per tutte le classi in stile VCL, *TObject*, ha un metodo *GetInterface* che permette di ottenere una classe *DelphiInterface* per qualsiasi interfaccia che è supportata da un'istanza di un oggetto. *TComponent* e *TInterfacedObject* implementano entrambi i metodi *IInterface* (*IUnknown*). Pertanto, qualsiasi discendente creato di *TComponent* o di *TInterfacedObject* eredita automaticamente il supporto per i metodi comuni di tutte le interfacce di Object Pascal. In Object Pascal, quando si crea un discendente dell'una o dell'altra di queste classi, è possibile supportare una nuova interfaccia implementando solo quei metodi introdotti dalla nuova interfaccia, e facendo affidamento su un'implementazione predefinita per i metodi ereditati di *IInterface*.

Sfortunatamente, poiché le firme dei metodi di *IUnknown* e di *IInterface* sono diverse tra C++Builder e le versioni utilizzate nelle classi VCL e CLX, le classi in stile VCL che vengono create non includono automaticamente il supporto per *IInterface* o per *IUnknown*, anche se sono derivate (direttamente o indirettamente) da *TComponent* o *TInterfacedObject*. Pertanto, per supportare qualsiasi interfaccia discendente da *IUnknown* o da *IInterface* che viene definita in C++, è sempre necessario aggiungere un'implementazione dei metodi di *IUnknown*.

Il modo più semplice per implementare i metodi di *IUnknown* in una classe che discende da *TComponent* o *TInterfacedObject* consiste nello sfruttare il supporto nativo per *IUnknown*, nonostante le differenze nelle firme delle funzioni. + sufficiente

aggiungere le implementazioni per le versioni in C++ dei metodi di *IUnknown*, delegando alle versioni basate su Object Pascal ereditate. Ad esempio:

```
virtual HRESULT __stdcall QueryInterface(const GUID& IID, void **Obj)
{
   return TInterfacedObject::QueryInterface(IID, (void *)Obj);
}
virtual ULONG __stdcall AddRef()
{
   return TInterfacedObject::_AddRef();
}
virtual ULONG __stdcall Release()
{
   return TInterfacedObject::_Release();
}
```

Se si aggiungono a un discendente di TinterfacedObject le tre implementazioni di metodo appena viste, la classe ottiene il completo supporto per *IUnknown* o *IInterface*.

Classi interfacciate e gestione del periodo di esistenza

I metodi di *IUnknown* delle classi interfaccia presentano alcune problematiche per il modo in cui vengono allocati e liberati gli oggetti che implementano l'interfaccia. Quando *IUnknown* è implementato da un oggetto COM, l'oggetto utilizza i metodi di *IUnknown* per tenere traccia di quanti riferimenti alla sua interfaccia sono utilizzati. Quando il contatore dei riferimenti passa a zero, l'oggetto viene liberato automaticamente. *TInterfacedObject* implementa i metodi di *IUnknown* per eseguire una gestione analoga del periodo di esistenza.

Questa interpretazione dei metodi di *IUnknown*, tuttavia, non è strettamente necessaria. L'implementazione predefinita dei metodi di *IUnknown* prevista da *TComponent*, ad esempio, ignora il conteggio dei riferimenti all'interfaccia, e pertanto il componente non viene liberato quando il numero dei riferimenti passa a zero. Ciò è dovuto al fatto che *TComponent* fa affidamento sull'oggetto specificato mediante la sua proprietà *Owner* per liberare le risorse.

Alcuni componenti utilizzano un ibrido di questi due modelli. Se la loro proprietà *Owner* è **NULL**, per la gestione del periodo di esistenza essi utilizzano il contatore dei riferimenti previsto dalla rispettiva interfaccia e vengono liberati quando quel contatore dei riferimenti arriva a zero. Nel caso abbiano un *Owner*, ignorano il contatore dei riferimenti e permettono al proprietario di liberarli. Si noti che per tali oggetti ibridi, come pure per qualsiasi altro oggetto che utilizza il contatore dei riferimenti per la gestione del periodo di esistenza, gli oggetti non vengono liberati automaticamente se l'applicazione crea l'oggetto ma non ottiene mai da esso un'interfaccia.

Identità e istanziazione di un oggetto

In C++ un'istanza di una classe è un oggetto reale. Questo può essere manipolato direttamente, ma è accessibile anche indirettamente tramite un riferimento o un puntatore.Per esempio, data una classe *CPP_class* di C++ con un costruttore che non

richiede alcun argomento, le seguenti sono tutte variabili di istanze valide di quella classe:

```
CPP_class by_value;// an object of type CPP_class
CPP_class& ref = by_value;// a reference to the object by_value, above
CPP_class* ptr = new CPP_class();// a pointer to an object of type CPP_class
```

Invece, in Object Pascal una variabile di tipo oggetto fa riferimento all'oggetto sempre in modo indiretto. La memoria viene allocata in modo dinamico per tutti gli oggetti. Per esempio, data una classe *OP_class* di Object Pascal

```
ref: OP_class;
ref := OP_class.Create;
```

ref è un "riferimento" ad un oggetto di tipo *OP_class*. Tradotto in codice C++Builder, diventerà

```
OP_class* ref = new OP_class;
```

Distinzione tra i riferimenti in C++ e in Object Pascal

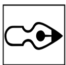
Spesso la documentazione tratta una variabile di istanza di una classe Object Pascal come se fosse un riferimento, ma ne descrive il comportamento come se fosse un puntatore. Ciò è dovuto al fatto che essa possiede le proprietà di entrambi. Un riferimento a oggetto di Object Pascal è simile ad un puntatore C++ con le seguenti eccezioni:

- Un riferimento Object Pascal viene dereferenziato in modo implicito (nel qual caso funziona anche come un riferimento C++).
- Un riferimento Object Pascal non ha un'aritmetica dei puntatori come un'operazione definita.

Confrontando un riferimento Object Pascal con un riferimento C++, si scoprono analogie e differenze. In entrambi i linguaggi i riferimenti sono dereferenziati in modo implicito, tuttavia,

- Un riferimento Object Pascal può essere ricollegato, mentre un riferimento C++ no.
- Un riferimento Object Pascal può essere **nil**, mentre un riferimento C++ deve fare riferimento a un oggetto valido.

Alcune delle decisioni di progettazione sottostanti la struttura VCL e CLX sono basate sull'uso di questo tipo di variabile di istanza. Un puntatore è il costrutto del linguaggio C++ più vicino ad un riferimento Object Pascal. Ne consegue che in C++Builder quasi tutti gli identificatori di oggetti VCL vengono tradotti in puntatori C++.



Il tipo di parametro **var** di Object Pascal è una corrispondenza molto simile ad un riferimento C++. Per maggiori informazioni sui parametri **var**, vedere "Parametri var" a pagina 13-17.

Copia di oggetti

A differenza di C++, Object Pascal non integra nel compilatore il supporto per eseguire la copia di un oggetto. Qui di seguito viene spiegato l'impatto di questa

differenza sugli operatori di assegnazione e sui costruttori di copia per le classi in stile VCL.

Operatori di assegnazione

L'operatore di assegnazione (:=) di Object Pascal non è un operatore di assegnazione di classe (operatore=()). L'operatore di assegnazione copia il riferimento, non l'oggetto. Nel codice seguente, *B* e *C* si riferiscono allo stesso oggetto:

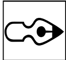
```
B, C: TButton;
B:= TButton.Create(ownerCtrl);
C:= B;
```

Questo esempio tradotto in C++Builder avrebbe il seguente codice:

```
TButton *B = NULL;
TButton *C = NULL;
B = new TButton(ownerCtrl);
C = B;// makes a copy of the pointer, not the object
```

Le classi in stile VCL in C++Builder seguono le regole del linguaggio Object Pascal per gli operatori di assegnazione. Ciò vuol dire che, nel codice seguente, le assegnazioni tra puntatori dereferenziati non sono valide in quanto esse tentano di copiare l'oggetto e non il puntatore:

```
TVCLStyleClass *p = new TVCLStyleClass;
TVCLStyleClass *q = new TVCLStyleClass;
*p = *q;// not allowed for VCL style class objects
```



Per le classi in stile VCL è sempre possibile usare la sintassi C++ per collegare un riferimento. Per esempio, il codice seguente è valido:

```
TVCLStyleClass *ptr = new TVCLStyleClass;
TVCLStyleClass &ref = *ptr;// OK for VCL style classes
```

Anche se ciò non equivale all'uso di un operatore di assegnazione, la sintassi è abbastanza simile perché valga la pena di riportarla in modo da poter effettuare un chiarimento e un confronto.

Costruttori di copia

Object Pascal non possiede costruttori di copia incorporati. Di conseguenza, in C++Builder le classi in stile VCL non hanno costruttori di copia incorporati. Il codice dell'esempio seguente cerca di creare un puntatore *TButton* con un costruttore di copia:

```
TButton *B = new TButton(ownerCtrl);
TButton *C = new TButton(*B);// not allowed for VCL style class objects
```

È opportuno non scrivere per le classi VCL e CLX codice che dipende da un costruttore di copia incorporato. Per creare una copia di un oggetto classe in stile VCL in C++Builder, si può scrivere il codice per una funzione membro che copia l'oggetto. In alternativa, i discendenti della classe *TPersistent* della VCL e della CLX possono ridefinire il metodo *Assign* per copiare i dati da un oggetto all'altro. Ciò normalmente viene fatto, per esempio, in classi grafiche come *TBitmap* e *TIcon* che contengono immagini di risorse. In conclusione, il modo di copiare un oggetto può essere determinato dal programmatore (da colui che scrive il componente); ma

occorre prestare attenzione al rischio che qualcuno dei metodi di copia usati in C++ standard non sia disponibili per le classi stile VCL.

Oggetti come argomenti di funzione

Come è già stato detto in precedenza, le variabili di istanza non sono indentiche in C++ e in Object Pascal. Queste differenze vanno tenute ben presenti quando si passano oggetti come argomenti alle funzioni. In C++, gli oggetti possono essere passati alle funzioni per valore, per riferimento e per puntatore. Quando in Object Pascal si passa per valore un oggetto a una funzione, occorre ricordare che l'oggetto è già un riferimento a un oggetto. Di fatto, è il riferimento che viene passato per valore, non l'oggetto reale. Non esiste alcun equivalente Object Pascal per il passaggio di un oggetto reale per valore in C++. Gli oggetti in stile VCL passati alle funzioni seguono le regole dell'Object Pascal.

Costruzione di oggetti per le classi VCL\CLX di C++Builder

C++ e Object Pascal costruiscono gli oggetti in modo differente. Questa sezione vuole essere una panoramica sull'argomento e presentare una descrizione di come C++Builder combina i due diversi approcci.

Costruzione di oggetti in C++

In C++ standard la sequenza di costruzione è: classi base virtuali, classi base e infine classi derivate. La sintassi di C++ usa l'elenco di inizializzazione del costruttore per chiamare i costruttori della classe base. Il tipo runtime dell'oggetto è quello della classe del costruttore attivo da chiamare. Il dispatch del metodo virtuale segue il tipo runtime dell'oggetto e cambia conseguentemente durante la costruzione.

Costruzione di oggetti in Object Pascal

In Object Pascal viene chiamato solo il costruttore per le classi istanziate; tuttavia, viene allocata la memoria per le classi base. La costruzione di ogni classe base immediata viene effettuata chiamando **inherited** nel costruttore della classe derivata corrispondente. Per convenzione, le classi della VCL e della CLX usano **inherited** per chiamare i costruttori della classe base (non vuota). Occorre tenere presente, comunque, che questo non è un requisito del linguaggio. Il tipo runtime dell'oggetto viene stabilito immediatamente uguale a quello della classe istanziata e non cambia quando vengono chiamati i costruttori della classe base. L'invio virtuale del metodo segue il tipo runtime dell'oggetto e pertanto non cambia durante la costruzione.

Costruzione di oggetti in C++Builder

Gli oggetti in stile VCL vengono costruiti come gli oggetti Object Pascal, ma usando la sintassi C++. Questo vuol dire che il metodo e l'ordine di chiamata dei costruttori della classe base segue la sintassi di C++ che usa l'elenco di inizializzazione per tutte le classi base non VCL e non CLX e il primo antenato VCL o CLX immediato. Questa classe base VCL o CLX è la prima classe che deve essere costruita. Opzionalmente, viene costruita una propria classe base usando **inherited** e seguendo il metodo Object Pascal. Pertanto, le classi base VCL e CLX vengono costruite nell'ordine

contrario a quello previsto in C++. Quindi, le classi base C++ vengono costruite completamente, dal più lontano antenato alla classe derivata. Il tipo runtime dell'oggetto e l'invio virtuale del metodo sono basati su Object Pascal.

La Figura 13.1 illustra la costruzione di un'istanza di una classe in stile VCL, *MyDerived*, derivata da *MyBase*, che è un diretto discendente di *TWinControl*. *MyDerived* e *MyBase* vengono implementate in C++. *TWinControl* è una classe VCL implementata in Object Pascal.

Figura 13.1 Sequenza di costruzione degli oggetti in stile VCL

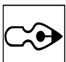
Inheritance

TObject
TPersistent
TComponent
TControl
TWinControl
C++/Object Pascal boundary
MyBase
MyDerived

Order of construction

TWinControl (constructor calls **inherited**)
TControl (constructor calls **inherited**)
TComponent
TPersistent (no constructor)
TObject (empty constructor)
MyBase
MyDerived

Si noti che, per un programmatore C++, la sequenza di costruzione potrebbe sembrare rovesciata, in quanto parte dall'antenato più esterno per un *TObject* di vere classi VCL e CLX, quindi costruisce *MyBase*, e infine costruisce la classe derivata.



TComponent non chiama **inherited** perché *TPersistent* non ha un costruttore. *TObject* ha un costruttore vuoto, quindi non viene chiamato. Se venissero chiamati questi

costruttori della classe, l'ordine seguirebbe il diagramma in cui queste classi appaiono in grigio.

I modelli di costruzione dell'oggetto in C++, in Object Pascal e in C++Builder sono riportati nella Tabella 13.1:

Tabella 13.1 Confronto dei modelli degli oggetti

C++	Object Pascal	C++Builder
Sequenza di costruzione		
Classi base virtuali, quindi classi base e infine la classe derivata.	Il costruttore della classe di cui è stata fatta un'istanza è il primo ed unico costruttore ad essere chiamato in modo automatico. Se vengono costruite altre classi, si parte dall'ultima per arrivare alla radice.	La classe base VCL o CLX più recente, quindi la costruzione segue il modello Object Pascal, e prosegue con il modello C++ (con l'unica differenza che le classi basi virtuali non sono consentite).
Metodo di chiamata dei costruttori della classe base		
Automaticamente, dall'elenco di inizializzazione del costruttore.	Opzionalmente, esplicitamente e in qualsiasi momento nel corpo del costruttore della classe derivata, usando la parola chiave **inherited**.	Automaticamente dall'elenco di inizializzazione del costruttore attraverso l'antenato più recente che è un costruttore della classe base VCL e CLX. Quindi secondo il metodo Object Pascal, chiamando i costruttori con **inherited**.
Tipo runtime dell'oggetto in fase di costruzione		
Cambia, riflettendo il tipo della classe attuale del costruttore.	Fissato immediatamente uguale a quello della classe istanziata.	Fissato immediatamente uguale a quello della classe istanziata.
Dispatch del metodo virtuale		
Cambia a seconda del tipo runtime dell'oggetto quando vengono chiamati i costruttori della classe base.	Segue il tipo runtime dell'oggetto, che rimare identico in tutte le chiamate a qualsiasi costruttore della classe.	Segue il tipo runtime dell'oggetto, che rimare identico in tutte le chiamate a qualsiasi costruttore della classe.

L'importanza di queste differenze viene descritta nella sezione seguente.

Chiamata dei metodi virtuali nei costruttori della classe base

I metodi virtuali richiamati dal corpo dei costruttori della classe base VCL o CLX, cioè, delle classi implementate in Object Pascal, vengono organizzati come in C++, secondo il tipo runtime dell'oggetto. Poiché C++Builder abbina immediatamente il modello Object Pascal di impostazione del tipo runtime di un oggetto con il modello C++ di costruzione delle classi base prima che la classe derivata venga costruita, chiamando i metodi virtuali dai costruttori della classe base per le classi stile VCL, si possono verificare effetti collaterali indesiderati. L'incidenza di ciò viene descritta

sotto, e poi viene illustrata in un esempio di una classe istanziata che viene derivata da almeno una classe base. In questa trattazione la classe istanziata viene considerata la classe derivata.

Modello Object Pascal

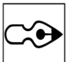
In Object Pascal i programmatori possono usare la parola chiave **inherited**, che fornisce la flessibilità necessaria per chiamare i costruttori della classe base da qualsiasi punto del corpo di un costruttore della classe derivata. Conseguentemente, se la classe derivata ridefinisce dei metodi virtuali che dipendono dall'impostazione dell'oggetto o dall'inizializzazione dei membri dati, ciò può succedere prima che vengano chiamati il costruttore della classe base e i metodi virtuali.

Modello C++

La sintassi C++ non ha la parola chiave **inherited** per chiamare il costruttore della classe base in qualsiasi momento durante la costruzione della classe derivata. Per il modello C++ l'uso di **inherited** non è necessario perché il tipo runtime dell'oggetto è quello della classe corrente che deve essere costruita, non quello della classe derivata. Perciò, i metodi virtuali chiamati sono quelli della classe attiva, non quelli della classe derivata. Conseguentemente, non è necessario inizializzare i membri dati o impostare l'oggetto della classe derivata prima che vengano chiamati questi metodi.

Modello C++Builder

In C++Builder gli oggetti in stile VCL hanno il tipo runtime della classe derivata in tutte le chiamate ai costruttori della classe base. Perciò, se il costruttore della classe base chiama un metodo virtuale, il metodo della classe derivata viene chiamato se la classe derivata lo ridefinisce. Se questo metodo virtuale dipende da qualcosa nell'elenco di inizializzazione o nel corpo del costruttore della classe derivata, il metodo viene chiamato prima che ciò accada. Per esempio, *CreateParams* è una funzione membro virtuale che viene chiamata indirettamente nel costruttore di *TWinControl*. Se si deriva una classe da *TWinControl* e si ridefinisce *CreateParams* in modo che dipenda da qualcosa nel costruttore, questo codice viene elaborato dopo che viene chiamata *CreateParams*. Questa situazione vale per qualsiasi classe derivata di una base. Si consideri una classe *C* derivata da *B*, che è derivata da A. Creando un'istanza di *C*, *A* chiamerebbe anche il metodo ridefinito di *B*, se *B* ridefinisce il metodo e *C* no.



Si deve tener presente che i metodi virtuali come *CreateParams* ovviamente non vengono chiamati nei costruttori, ma vengono chiamati indirettamente.

Esempio: chiamata dei metodi virtuali

L'esempio seguente confronta le classi C++ e quelle in stile VCL che hanno sostituito i metodi virtuali. Questo esempio illustra come vengono risolte in entrambi i casi le chiamate a quei metodi virtuali dai costruttori della classe base. *MyBase* e *MyDerived* sono classi standard di C++. *MyVCLBase* e *MyVCLDerived* sono classi in stile VCL derivate da *TObject*. Il metodo virtuale *what_am_I()* viene ridefinito in entrambe le classi derivate, ma viene chiamato *solo* nei costruttori della classe base, non nei costruttori della classe derivata.

```
#include <sysutils.hpp>
#include <iostream.h>
// non-VCL style classes
class MyBase {
public:
   MyBase() { what_am_I(); }
   virtual void what_am_I() { cout << "I am a base" << endl; }
};
class MyDerived : public MyBase {
public:
   virtual void what_am_I() { cout << "I am a derived" << endl; }
};
// VCL style classes
class MyVCLBase : public TObject {
public:
   __fastcall MyVCLBase() { what_am_I(); }
   virtual void __fastcall  what_am_I() { cout << "I am a base" << endl; }
};
class MyVCLDerived : public MyVCLBase {
public:
   virtual void __fastcall  what_am_I() { cout << "I am a derived" << endl; }
};
int main(void)
{
   MyDerived d;// instantiation of the C++ class
   MyVCLDerived *pvd = new MyVCLDerived;// instantiation of the VCL style class
   return 0;
}
```

L'output di questo esempio è

```
I am a base
I am a derived
```

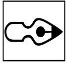
a causa della differenza nei tipi runtime di *MyDerived* e *MyVCLDerived* durante le chiamate ai rispettivi costruttori della classe base.

Inizializzazione del costruttore dei membri dati per le funzioni virtuali

Poiché i membri dati possono essere usati in funzioni virtuali, è importante capire quando e come vengono inizializzati. In Object Pascal tutti i dati non inizializzati vengono inizializzati a zero. Questo vale, per esempio, per le classi base i cui costruttori non vengono chiamati con **inherited**. In C++ standard non c'è alcuna garanzia sul valore dei membri dati non inizializzati. I seguenti tipi di membri dati di classe devono essere inizializzati nell'elenco di inizializzazione del costruttore della classe:

- riferimenti
- membri dati senza costruttori predefiniti

Ciò nonostante, il valore di questi membri dati, o di quelli inizializzati nel corpo del costruttore, non è definito quando vengono chiamati i costruttori della classe base. In C++Builder la memoria per le classi in stile VCL è inizializzata a zero.



Tecnicamente, è la memoria della classe VCL o CLX che è zero, ovvero sono i bit a zero, mentre i valori non sono in realtà definiti. Per esempio, un riferimento è zero.

Una funzione virtuale che fa affidamento sul valore delle variabili membro inizializzate nel corpo del costruttore o nell'elenco di inizializzazione può comportarsi come se le variabili fossero inizializzate a zero. Questo si verifica perché il costruttore della classe base viene chiamato prima che l'elenco di inizializzazione venga elaborato o che il corpo del costruttore venga immesso. L'esempio seguente illustra ciò che succede:

```
#include <sysutils.hpp>

class Base : public TObject {
public:
   __fastcall Base() { init(); }
   virtual void __fastcall  init() { }
};

class Derived : public Base {
public:
   Derived(int nz) : not_zero(nz) { }
   virtual void __fastcall  init()
   {
      if (not_zero == 0)
         throw Exception("not_zero is zero!");
   }
private:
   int not_zero;
};

int main(void)
{
   Derived *d42 = new Derived(42);
   return 0;
}
```

Questo esempio presenta un'eccezione nel costruttore di *Base*. Poiché *Base* viene costruita prima di *Derived, not_zero*, non è stata ancora inizializzata con il valore di 42 passato al costruttore. Si tenga presente che non è possibile inizializzare i membri dati della classe in stile VCL prima che vengano chiamati i costruttori della sua classe base.

Distruzione di oggetti

Due meccanismi, riguardanti la distruzione degli oggetti, funzionano differentemente in C++ e in Object Pascal. Questi sono:

- distruttori chiamati a causa delle eccezioni presentate dai costruttori
- i metodi virtuali chiamati dai distruttori

Le classi in stile VCL abbinano i metodi di questi due linguaggi. Gli aspetti vengono discussi qui di seguito.

Eccezioni sollevate dai costruttori

Se si verifica un'eccezione durante la costruzione dell'oggetto, i distruttori vengono chiamati in modo differente in C++ e in Object Pascal. Si prenda come esempio una classe *C*, derivata da una classe *B*, che è derivata da una classe *A*:

```
class A
{
   // body
};

class B: public A
{
   // body
};

class C: public B
{
   // body
};
```

Supponendo che si sia verificata un'eccezione nel costruttore della classe *B* durante la costruzione di un'istanza di *C*, questi sarebbero i risultati in C++, in Object Pascal e in classi in stile VCL:

- In C++ prima vengono chiamati i distruttori per tutti i membri dati dell'oggetto completamente costruiti di *B*, quindi viene chiamato il distruttore di *A*, e successivamente vengono chiamati i distruttori per tutti i membri dati dell'oggetto completamente costruiti di *A*. Comunque, i distruttori per *B* e *C* non vengono chiamati.
- In Object Pascal solo il distruttore della classe istanziata viene chiamato automaticamente. Questo è il distruttore per *C*. Come avviene con i costruttori, è esclusivamente responsabilità del programmatore chiamare **inherited** nei distruttori. In questo esempio, se si suppone che tutti i distruttori chiamano **inherited**, allora i distruttori per *C*, *B* e *A* vengono chiamati in questo ordine. Inoltre, indipendentemente dal fatto che **inherited** sia già stata o meno chiamata nel costruttore di *B* prima che si verificasse l'eccezione, il distruttore di *A* viene chiamato perché **inherited** è stato chiamato nel distruttore di *B*. La chiamata al distruttore per *A* è indipendente dal fatto che il suo costruttore sia stato effettivamente chiamato. Inoltre, fattore più importante, poiché di solito i costruttori chiamano immediatamente **inherited**, il distruttore per *C* viene chiamato indipendentemente dal fatto che il corpo del suo costruttore sia stato o no completamente eseguito.
- Nelle classi in stile VCL le basi della VCL o della CLX (implementate in Object Pascal) seguono il metodo Object Pascal di chiamata dei distruttori. Le classi derivate (implementate in C++) non seguono esattamente nessuno dei metodi del linguaggio. Succede invece che vengono chiamati tutti i distruttori; ma i corpi di quelli che non saranno stati chiamati secondo le regole del linguaggio C++, non vengono immessi.

Le classi implementate in Object Pascal, pertanto, forniscono un'opportunità per elaborare qualsiasi codice di cleanup scritto nel corpo del distruttore. Ciò è vero anche per il codice che libera memoria per i sotto-oggetti (membri dati che sono

oggetti) che vengono costruiti prima che si verifichi un'eccezione del costruttore. Si deve tenere presente che, per le classi in stile VCL, il codice di cleanup può non essere elaborato per la classe istanziata o per le sue basi implementate con C++, anche se vengono chiamati i distruttori.

Per maggiori informazioni sulla gestione delle eccezioni in C++Builder, consultare "Gestione delle eccezioni VCL/CLX" a pagina 12-15.

Metodi virtuali chiamati dai distruttori

L'invio dei metodi virtuali nei distruttori segue lo stesso modello definito per i costruttori. Questo significa che per le classi in stile VCL, la classe derivata viene distrutta per prima, ma il tipo runtime dell'oggetto rimane quello della classe derivata in tutte le successive chiamate ai distruttori della classe base. Pertanto, se i metodi virtuali vengono chiamati nei distruttori della classe base VCL o CLx, c'è il rischio che l'invio riguardi una classe che è già stata distrutta.

AfterConstruction e BeforeDestruction

TObject propone due metodi virtuali, *BeforeDestruction* e *AfterConstruction*, che consentono ai programmatori di scrivere del codice che viene elaborato rispettivamente prima e dopo la distruzione e la creazione degli oggetti. *AfterConstruction* viene chiamato dopo la chiamata dell'ultimo costruttore. *BeforeDestruction* viene chiamato prima che venga chiamato il primo distruttore. Questi metodi sono pubblici e vengono chiamati automaticamente.

Funzioni virtuali di classe

Object Pascal ha il concetto di funzione virtuale di classe.L'analogo C++ sarebbe una funzione virtuale statica, se fosse possibile; ma C++ non ha alcun corrispondente di questo tipo di funzione. Queste funzioni vengono chiamate con sicurezza all'interno della VCL e della CLX. Tuttavia, in C++Builder non verrà mai chiamata una funzione di questo tipo. + possibile identificare queste funzioni nei file header in quanto sono precedute dal seguente commento:

```
/* virtual class method */
```

Supporto per i tipi di dati Object Pascal e per le strutture del linguaggio

Per supportare la VCL e la CLX, C++Builder implementa, traduce o mette comunque in relazione la maggior parte dei tipi di dati, dei costrutti e delle strutture del linguaggio Object Pascal con quelli del linguaggio C++. Ciò viene realizzato nei seguenti modi:

- Typedefs per i tipi C++ nativi
- Classes, strutture e modelli di classe
- Equivalenti del linguaggio C++

- Macro
- Parole chiave che sono estensioni del linguaggio ANSI standard

Non tutti gli aspetti del linguaggio Object Pascal coincidono perfettamente con quelli del C++. A volte, l'uso di queste parti del linguaggio può far sì che l'applicazione abbia dei comportamenti inattesi. Ad esempio:

- Alcuni tipi esistono in Object Pascal e in C++, ma vengono definiti in modo diverso. Essi possono richiedere una particolare attenzione quando il codice viene condiviso dai due linguaggi.
- Alcune estensioni sono state aggiunte a Object Pascal con lo scopo di supportare C++Builder. Occasionalmente esse possono avere una notevole incidenza a livello di interoperatività.
- I tipi e i costrutti del linguaggio Object Pascal che hanno alcun corrispondente nel linguaggio C++ saranno evitati in C++Builder quando il codice viene condiviso tra questi linguaggi.

La sezione successiva presenta un riepilogo dei modi in cui C++Builder implementa il linguaggio Object Pascal, e alcuni suggerimenti su quando occorre usare una certa prudenza.

Typedef

La maggior parte dei tipi di dati intrinseci Object Pascal viene implementata in C++Builder usando un **typedef** per un tipo C++ nativo. Questi si trovano in sysmac.h. Ogni qual volta è possibile, è opportuno usare il tipo C++ nativo, anziché il tipo Object Pascal.

Classi che supportano il linguaggio Object Pascal

Alcuni tipi di dati e costrutti del linguaggio Object Pascal che non hanno un equivalente incorporato in C++ vengono implementati come classi o strutture. I modelli di classe vengono usati anche per implementare i tipi di dati e i costrutti del linguaggio Object Pascal, come **set**, da cui può essere dichiarato uno specifico tipo. Le relative dichiarazioni si trovano nei seguenti file header:

- dstring.h
- wstring.h
- sysclass.h
- syscomp.h
- syscurr.h
- sysdyn.h
- sysopen.h
- sysset.h
- systdate.h
- systobj.h
- systvar.h
- sysvari.h

Le classi implementate in questi file header sono state create per supportare i tipi nativi usati nelle routine Object Pascal. Esse sono destinate ad essere usate quando queste routine vengono chiamate nel codice basato su VCL o CLX.

Equivalenti del linguaggio C++ presenti nel linguaggio Object Pascal

I parametri **var** e untyped di Object Pascal non sono nativi di C++. Entrambi hanno degli equivalenti nel linguaggio C++ che vengono usati in C++Builder.

Parametri var

C++ e Object Pascal condividono la logica del “passaggio per riferimento”. Questi sono argomenti modificabili. In Object Pascal vengono chiamati parametri **var**. La sintassi per le funzioni che richiedono un parametro **var** è

```
procedure myFunc(var x : Integer);
```

In C++ questi tipi di parametro devono essere passati per riferimento:

```
void myFunc(int& x);
```

I riferimenti e i puntatori di C++ possono essere usati per modificare l’oggetto. Tuttavia, un riferimento è il corrispondente più vicino ad un parametro **var** in quanto, a differenza di un puntatore, un riferimento non può essere rebound e un parametro **var** non può essere assegnato una seconda volta; anche se è possibile cambiare il valore di ciò a cui fa riferimento.

Parametri senza tipo

Object Pascal consente l’uso di parametri di tipo non specificato. Questi parametri vengono passati alle funzioni senza alcun tipo definito. La funzione ricevente deve ricondurre il parametro ad un tipo noto prima di usarlo. C++Builder interpreta i parametri di tipo non specificato come puntatori a vuoto (*void **). La funzione ricevente deve ricondurre il puntatore vuoto ad un puntatore del tipo desiderato. Questo è un esempio:

```
int myfunc(void* MyName)
{
   // Cast the pointer to the correct type; then dereference it.
   int* pi = static_cast<int*>(MyName);
   return 1 + *pi;
}
```

Open array

Object Pascal ha un costrutto “open array” che permette di passare a una funzione un array di dimensioni non specificate. Mentre in C++ non esiste alcun supporto diretto per questo tipo, una funzione Object Pascal che ha un parametro array aperto può essere chiamata passando esplicitamente il puntatore al primo elemento di un array, e il valore all’ultimo indice (il numero di elementi dell’array meno uno). Per esempio, la funzione *Mean* in math.hpp ha questa dichiarazione in Object Pascal:

```
function Mean(Data: array of Double): Extended;
```

La dichiarazione in C++ è

```
Extended __fastcall Mean(const double * Data, const int Data_Size);
```

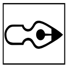
Il codice seguente illustra la chiamata della funzione *Mean* da C++:

```
double d[] = { 3.1, 4.4, 5.6 };

// explicitly specifying last index
long double x = Mean(d, 2);

// better: use sizeof to ensure that the correct value is passed
long double y = Mean(d, (sizeof(d) / sizeof(d[0])) - 1);

// use macro in sysopen.h
long double z = Mean(d, ARRAYSIZE(d) - 1);
```



In casi simili all'esempio sopra riportato, ma dove la funzione Object Pascal aspetta un parametro **var**, i parametri di dichiarazione di funzione C++ non saranno **const**.

Calcolo del numero di elementi

Quando si usa *sizeof()*, la macro ARRAYSIZE o la macro EXISTINGARRAY per calcolare il numero di elementi di un array, occorre prestare attenzione a non usare invece un puntatore all'array. Passare invece il nome dello stesso array:

```
double d[] = { 3.1, 4.4, 5.6 };
int n = ARRAYSIZE(d); // sizeof(d)/sizeof(d[0]) => 24/8 => 3

double *pd = d;
int m = ARRAYSIZE(pd); // sizeof(pd)/sizeof(pd[0]) => 4/8 => 0 => Error!
```

Prendere il "sizeof" di un array non equivale a prendere il "sizeof" di un puntatore. Per esempio, data la seguente dichiarazione,

```
double d[3];
double *p = d;
```

che assume le dimensioni dell'array come mostrato

```
sizeof(d)/sizeof d[0]
```

non equivale a calcolare le dimensioni del puntatore:

```
sizeof(p)/sizeof(p[0])
```

Questo esempio e quelli seguenti usano la macro ARRAYSIZE anziché l'operatore *sizeof()*. Per maggiori informazioni sulla macro ARRAYSIZE, consultare la Guida in linea.

Array temporanei

Object Pascal fornisce il supporto per il passaggio di array aperti temporanei senza nome alle funzioni. In C++ non esiste alcuna sintassi che svolga questa operazione. Tuttavia, poiché le definizioni di variabili possono essere mischiate con altre istruzioni, un modo per risolvere il problema è quello di fornire semplicemente la variabile con un nome.

Object Pascal:

```
Result := Mean([3.1, 4.4, 5.6]);
```

C++, usando un array “temporaneo” con nome:

```
double d[] = { 3.1, 4.4, 5.6 };
return Mean(d, ARRAYSIZE(d) - 1);
```

Per restringere il campo d’azione dell’array “temporaneo” con nome ed evitare conflitti con altre variabili locali, è consigliabile aprire un nuovo campo d’azione:

```
long double x;
{
   double d[] = { 4.4, 333.1, 0.0 };
   x = Mean(d, ARRAYSIZE(d) - 1);
}
```

Per un’altra soluzione, vedere “Macro OPENARRAY” a pagina 13-19.

Array of const

Object Pascal supporta un costrutto del linguaggio chiamato **array of const**. Questo tipo di argomento equivale ad acquisire un open array di *TVarRec* per valore.

Il seguente è un segmento di codice Object Pascal dichiarato per accettare un **array of const**:

```
function Format(const Format: string; Args: array of const ): string;
```

In C++ il prototipo è

```
AnsiString __fastcall Format(const AnsiString Format,
      TVarRec const *Args, const int Args_Size);
```

La funzione viene chiamata proprio come qualsiasi altra funzione che acquisisce un open array:

```
void show_error (int error_code, AnsiString const &error_msg)
{
   TVarRec v[] = { error_code, error_msg };
   ShowMessage(Format("%d: %s", v, ARRAYSIZE(v) - 1));
}
```

Macro OPENARRAY

La macro OPENARRAY definita in sysopen.h può essere usata come alternativa ad una variabile con nome per passare un open array temporaneo ad una funzione che acquisisce un open array per valore. L’uso della macro assume questo formato

```
OPENARRAY(T, (value1, value2, value3)) // up to 19 values
```

dove *T* è il tipo open array da costruire e i parametri *value* sono utilizati per compilare l’array. Le parentesi per racchiudere gli argomenti dei *valori* sono necessarie. Ad esempio:

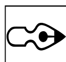
```
void show_error(int error_code, AnsiString const &error_msg)
{
   ShowMessage(Format("%d: %s", OPENARRAY(TVarRec, (error_code, error_msg))));
}
```

Con la macro OPENARRAY possono essere passati fino a 19 valori. Se fosse necessario un array più grande, occorrerebbe definire una variabile esplicita. Inoltre, usando la questa macro, si paga un ulteriore piccolo costo in fase di esecuzione, dovuto sia alla spazio richiesto per l’allocazione dell’array sottostante sia alla ulteriore copia di ciascun valore.

Macro EXISTINGARRAY

La macro EXISTINGARRAY definita in sysopen.h può essere usata per passare un array esistente al posto di un open array. L’uso della macro assume questo formato

```
long double Mean(const double *Data, const int Data_Size);
double d[] = { 3.1, 3.14159, 2.17128 };
Mean(EXISTINGARRAY (d));
```



La sezione “Calcolo del numero di elementi” a pagina 13-18 vale anche per la macro EXISTINGARRAY.

Funzioni C++ che accettano argomenti di tipo open array

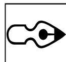
Quando si scrive una funzione C++ a cui verrà passato un array aperto da Object Pascal, è importante conservare esplicitamente la semantica del “passaggio per valore”. In particolare, se la dichiarazione per la funzione corrisponde al “passaggio per valore”, bisogna accertarsi di aver copiato esplicitamente tutti gli elementi prima di modificarli. In Object Pascal un array aperto è un tipo incorporato e può essere passato per valore. In C++ il tipo array aperto viene implementato usando un puntatore, il quale modifica l’array originale, a meno che non ne venga fatta una copia locale.

Tipi definiti in altro modo

I tipi che in Object Pascal e C++ vengono definiti in modo diverso normalmente non sono causa di preoccupazioni. I rari casi in cui sono problematici risultano difficilmente individuabili. Per tale motivo, questi tipi vengono trattati in questa sezione.

Tipi di dati booleani

In Object Pascal il valore *True* per i tipi di dati *ByteBool*, *WordBool* e *LongBool* viene rappresentato con –1. *False* viene rappresentato con 0.



Il tipo di dati Boolean rimane costante (true = 1, false = 0)).

Mentre il tipo bool di C++ converte correttamente questi tipi di Object Pascal, nascono problemi quando vengono condivise una funzione WinAPI o una qualsiasi altra funzione che usa un tipo BOOL di Windows rappresentato con 1. Se viene passato un valore ad un parametro di tipo BOOL, esso vale –1 in Object Pascal e 1 in C++. Pertanto, se tra questi due linguaggi viene condiviso del codice, qualsiasi confronto dei due identificatori può risultare errato, a meno che non siano entrambi 0 (*False*, false). Per aggirare l’ostacolo, si può usare il seguente metodo di confronto:

```
!A == !B;
```

La Tabella 13.2 mostra i risultati dell'uso di questo metodo di confronto dell'uguaglianza:

Tabella 13.2 Confronto dell'uguaglianza !A == !B di variabili BOOL

Object Pascal	C++	!A == !B
0 (False)	0 (false)	!0 == !0 (TRUE)
0 (False)	1 (true)	!0 == !1 (FALSE)
–1 (True)	0 (false)	!–1 == !0 (FALSE)
–1 (True)	1 (true)	!–1 == !1 (TRUE)

Con questo metodo di confronto qualunque serie di valori sarà calcolata correttamente.

Tipi di dati char

Il tipo char in C++ è con segno, mentre in Object Pascal è senza segno. + molto raro che si verifichi una situazione in cui questa differenza potrebbe diventare un problema quando viene condiviso del codice.

Interfacce Delphi

l compilatore Object Pascal gestisce automaticamente molti dettagli durante le attività con le interfacce. Incrementa automaticamente il contatore dei riferimenti di un'interfaccia quando il codice dell'applicazione acquisisce un puntatore all'interfaccia e decrementa tale contatore quando l'interfaccia esce dal campo di validità.

In C++Builder, la classe modello *DelphiInterface* fornisce alle classi interfaccia di C++ solo una parte di tali automatismi. Per le proprietà e i metodi nella VCL e nella CLX che utilizzano i tipi interfaccia in Object Pascal, i wrapper di C++ utilizzano una *DelphiInterface* che è costruita utilizzando la classe interfaccia sottostante.

Il costruttore *DelphiInterface*, il costruttore copia, l'operatore di assegnazione e il distruttore incrementano o decrementano all'occorrenza il contatore dei riferimenti. Tuttavia, *DelphiInterface* non è così conveniente quanto il supporto offerto dal compilatore per le interfacce in Object Pascal. Altri operatori che forniscono l'accesso al puntatore all'interfaccia sottostante non gestiscono il conteggio dei riferimenti perché la classe non può sempre dire quando è opportuno farlo. Potrebbe essere necessario chiamare esplicitamente *AddRef* o *Release* per essere certi che il conteggio dei riferimenti sia corretto.

Stringhe di risorse

Se nel codice è presente una unit Pascal che usa stringhe di risorse, il compilatore Pascal (DCC32) genera una variabile globale e una corrispondente macro del preprocessore per ciascuna stringa di risorse quando genera il file header. Le macro vengono usate per caricare automaticamente le stringhe di risorse, e sono destinate

ad essere usate nel codice C++ dovunque si fa riferimento alla stringa di risorse. Per esempio, la sezione **resourcestring** nel codice Object Pascal può contenere

```
unit borrowed;
interface
resourcestring
   Warning = 'Be careful when accessing string resources.';
implementation
begin
end.
```

Il codice corrispondente generato dal compilatore Pascal per C++Builder sarà

```
extern PACKAGE System::Resource ResourceString _Warning;
#define Borrowed_Warning System::LoadResourceString(&Borrowed::_Warning)
```

Questo consente di usare le stringhe di risorse Object Pascal esportate senza dover chiamare in modo esplicito *LoadResourceString*.

Parametri predefiniti

La versione più recente del compilatore Pascal accetta parametri predefiniti per i costruttori per compatibilità con C++. A differenza di C++, i costruttori Object Pascal possono però avere stesso numero e gli stessi tipi di parametri, poiché ad essi viene attribuito un nome univoco. In tali casi nei costruttori Object Pascal vengono usati parametri fittizi per distinguerli quando vengono generati i file header di C++. Per esempio, per una classe chiamata *TInCompatible*, i costruttori Object Pascal potrebbero essere

```
constructor Create(AOwner: TComponent);
constructor CreateNew(AOwner: TComponent);
```

che porterebbero, senza parametri predefiniti, al seguente codice ambiguo in C++ per entrambi i costruttori:

```
__fastcall TInCompatible(Classes::TComponent* Owner);// C++ version of the Pascal Create
constructor

__fastcall TInCompatible(Classes::TComponent* Owner);// C++ version of the Pascal CreateNew
constructor
```

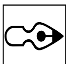
Tuttavia, usando parametri predefiniti, per una classe chiamata *TCompatible*, i costruttori Object Pascal sono

```
constructor Create(AOwner: TComponent);
constructor CreateNew(AOwner: TComponent; Dummy: Integer = 0);
```

Essi portano al seguente codice non ambiguo in C++Builder:

```
__fastcall TCompatible(Classes::TComponent* Owner);// C++ version of the Pascal Create
constructor

__fastcall TCompatible(Classes::TComponent* Owner, int Dummy);// C++ version of the Pascal
CreateNew constructor
```



Il problema principale con i parametri predefiniti è che DCC32 rimuove il valore predefinito del parametro predefinito. Un errore in tale operazione può generare

ambiguità qualora non ci siano parametri predefiniti. Bisogna tenerne conto quando si usano classi VCL o CLX oppure componenti di produttori terzi.

Informazioni di tipo runtime

Object Pascal ha costrutti di linguaggio che hanno a che fare con RTTI. Alcuni hanno equivalenti in C++. Essi vengono elencati nella Tabella 13.3:

Tabella 13.3 Esempi di correlazioni RTTI da Object Pascal a C++

RTTI Object Pascal	RTTI C++
if Sender **is** TButton...	**if** (dynamic_cast <TButton*> (Sender) // dynamic_cast restituisce **NULL** in caso di errore.
b := Sender **as** TButton; (* genera un'eccezione in caso di errore *)	TButton& ref_b = dynamic_cast <TButton&> (*Sender) // genera un'eccezione in caso di errore.
ShowMessage(Sender.ClassName);	ShowMessage(typeid(*Sender).name());

Nella Tabella 13.3, *ClassName* è un metodo *TObject* che restituisce una stringa contenente il nome del tipo attuale dell'oggetto, indipendentemente dal tipo di variabile dichiarata. Gli altri metodi RTTI introdotti in *TObject* non hanno equivalenti in C++. Essi sono tutti pubblici e sono elencati qui di seguito:

- *ClassInfo* restituisce un puntatore alla tabella contenente le informazioni di tipo runtime (RTTI) del tipo di oggetto.
- *ClassNameIs* determina se un oggetto è di uno specifico tipo.
- *ClassParent* restituisce il tipo dell'antenato diretto della classe. Nel caso di *TObject* *ClassParent* restituisce nil perché *TObject* non non ha alcun parente. Viene usato dagli operatori **is** e **as**, e dal metodo *InheritsFrom*.
- *ClassType* determina dinamicamente il tipo attuale di un oggetto. Viene usato internamente dagli operatori **is** e **as** di Object Pascal.
- *FieldAddress* usa RTTI per ottenere l'indirizzo di un campo pubblicato. Viene usato internamente dal sistema steaming.
- *InheritsFrom* determina la relazione tra due oggetti. Viene usato internamente dagli operatori **is** e **as** di Object Pascal.
- *MethodAddress* usa RTTI per trovare l'indirizzo di un metodo. Viene usato internamente dal sistema steaming.

Alcuni di questi metodi di *TObject* sono destinati principalmente all'uso interno da parte del compilatore o del sistema streaming. Per maggiori informazioni, consultare la Guida in linea.

Tipi non mappati

Tipi real a 6 byte

Il vecchio formato a virgola mobile a 6 byte di Object Pascal ora viene chiamato *Real48*. Il vecchio tipo *Real* ora è un double. C++ non ha un equivalente per il tipo *Real48*. Conseguentemente, non si deve usare del codice Object Pascal che includa questo tipo con il codice C++. Altrimenti, il generatore del file header genererà un messaggio.

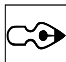
Arrays come tipi restituiti da funzioni

In Object Pascal una funzione può richiedere come argomento, o restituire come tipo, un array. Per esempio, la sintassi per una funzione *GetLine* che restituisce un array di 80 caratteri è

```
type
  Line_Data = array[0..79] of char;
function GetLine: Line_Data;
```

C++ non ha un equivalente di questa funzione. In C++ gli array non sono consentiti come tipi restituiti da funzioni. E tantomeno C++ accetta array come tipo di argomento di una funzione.

È bene tenere presente che, sebbene la VCL e la CLX non abbiano alcuna proprietà che sia un array, il linguaggio Object Pascal lo consente. Dal momento che le proprietà possono usare i metodi di lettura e di scrittura Get e *Set*, che richiedono e restituiscono valori del tipo della proprietà, in C++Builder non è possibile avere una proprietà di tipo array.



Le proprietà array, che sono valide anche in Object Pascal, non sono un problema in C++ perché il metodo Get prende un valore di indice come parametro, mentre il metodo Set restituisce un oggetto del tipo contenuto dall'array. Per maggiori informazioni sulle proprietà array, consultare "Creazione di proprietà array" a pagina 47-8.

Estensioni di parole chiave

Questa sezione descrive le estensioni di parole chiave conformi con lo standard ANSI, implementate in C++Builder per supportare la VCL e la CLX. Per un elenco completo delle parole chiave e delle relative estensioni in C++Builder, consultare la Guida in linea.

__classid

L'operatore **__classid** viene usato dal compilatore per generare un puntatore a vtable per il *classname* specificato. Questo operatore viene usato per ottenere la classe meta da una classe.

Sintassi

```
__classid(classname)
```

Per esempio, **__classid** viene usato per la registrazione di editor di proprietà, di componenti e di classi, e con il metodo *InheritsFrom* di *TObject*. Il codice seguente

illustra l'uso di **__classid** per la creazione di un nuovo componente derivato da *TWinControl*:

```
namespace Ywndctrl
{
  void __fastcall PACKAGE Register()
  {
    TComponentClass classes[1] = {__classid(MyWndCtrl)};
    RegisterComponents("Additional", classes, 0);
  }
}
```

__closure

La parola chiave **__closure** serve per dichiarare uno speciale tipo di puntatore ad una funzione membro. Nel linguaggio C++ standard, il solo modo per ottenere un puntatore a una funzione membro consiste nell'utilizzare il nome del membro per esteso, come mostrato nell'esempio seguente:

```
class base
{
  public:
    void func(int x) { };
};

typedef void (base::* pBaseMember)(int);

int main(int argc, char* argv[])
{
  base        baseObject;
  pBaseMember m = &base::func; // Get pointer to member 'func'

  // Call 'func' through the pointer to member
  (baseObject.*m)(17);
  return 0;
}
```

Tuttavia, non è possibile assegnare un puntatore a un membro di una classe derivata a un puntatore a un membro di una classe di base. Questa regola (chiamata *contravariance*) è illustrata nell'esempio seguente:

```
class derived: public base
{
  public:
    void new_func(int i) { };
};

int main(int argc, char* argv[])
{
  derived        derivedObject;
  pBaseMember m = &derived::new_func; // ILLEGAL

  return 0;
}
```

L'estensione della parola chiave **__closure** permette di aggirare questa limitazione e di fare qualcosa in più. Utilizzando un closure è possibile ottenere un puntatore a funzione membro di un oggetto (cioè una particolare istanza di una classe). L'oggetto può essere *qualsiasi* oggetto, indipendentemente dalla sua gerarchia di eredità. Il puntatore **this** dell'oggetto viene utilizzato automaticamente quando si chiama la funzione membro mediante il closure. L'esempio seguente mostra come dichiarare e utilizzare un closure. Si presume che le classi *base* e *derivate* fornite in precedenza siano state definite.

```
int main(int argc, char* argv[])
{
  derived        derivedObject;
  void (__closure *derivedClosure)(int);

     derivedClosure = derivedObject.new_func; // Get a pointer to the 'new_func' member.
                                              // Note the closure is associated with the
                                              // particular object, 'derivedObject'.
     derivedClosure(3);  // Call 'new_func' through the closure.
     return 0;
}
```

I closure funzionano inoltre con puntatori a oggetti, come illustrato nell'esempio seguente:

```
void func1(base *pObj)
{
  // A closure taking an int argument and returning void.
  void ( __closure *myClosure )(int);

  // Initialize the closure.
  myClosure = pObj->func;

  // Use the closure to call the member function.
  myClosure(1);
  return;
}

int main(int argc, char* argv[])
{
  derived        derivedObject;
  void (__closure *derivedClosure)(int);

     derivedClosure = derivedObject.new_func; // Same as before...
     derivedClosure(3);

     // We can use pointers to initialize a closure, too.
     // We can also get a pointer to the 'func' member function
     // in the base class.
     func1(&derivedObject);
     return 0;
}
```

Notare che si sta passando un puntatore a un'istanza della classe *derivata* e lo si sta utilizzando per ottenere un puntatore a una funzione membro nella classe *base* – una cosa che il linguaggio C++ standard non consente di fare.

I closure sono una parte fondamentale dell'ambiente RAD di C++ Builder. Essi danno la capacità di assegnare un gestore di evento nell'Object Inspector. Ad esempio, un oggetto *TButton* ha un evento di nome *OnClick*. Nella classe *TButton*, l'evento *OnClick* è una proprietà che utilizza nella sua dichiarazione l'estensione della parola chiave **__closure**. La parola chiave **__closure** permette di assegnare alla proprietà una funzione membro di un'altra classe (di solito una funzione membro in un oggetto *TForm*). Quando si colloca un oggetto *TButton* su una scheda e quindi si crea un gestore per l'evento *OnClick* del pulsante, C++Builder crea una funzione membro nel *TForm* genitore del pulsante e assegna quella funzione membro all'evento *OnClick* di *TButton*. In questo modo, il gestore di evento viene associato a quella particolare istanza di *TButton* e a nessun altro.

Per ulteriori informazioni su eventi e closure, vedere il Chapter 48, "Creazione di eventi".

__property

La parola chiave **__property** dichiara un attributo di una classe. Le proprietà appaiono al programmatore esattamente come qualsiasi altro attributo (campo) di una classe. Tuttavia, come la sua controparte Object Pascal, la parola chiave **__property** di C++Builder aggiunge molte altre funzionalità altre alla semplice capacità di esaminare e modificare il valore dell'attributo. Poiché gli attributi della proprietà controllano completamente l'accesso alla proprietà, non esistono limitazionni sul modo in cui si implementa la proprietà all'interno della stessa classe.

Sintassi

```
__property type propertyName[index1Type index1][indexNType indexN] = { attributes };
```

in cui

- *type* è un tipo di dati intrinseco o dichiarato in precedenza.
- *propertyName* è un qualsiasi identificativo valido.
- *indexNType* è un tipo di dati intrinseco o dichiarato in precedenza.
- *indexN* è il nome di un parametro indice che sarà passato alle funzioni **read** e **write** della proprietà.
- *attributes* è una sequenza separata da virgole di **read**, **write**, **stored**, **default** (o **nodefault**), o **index**.

I parametri *indexN* fra parentesi quadre sono facoltativi. Se presenti, dichiarano una proprietà array. I parametri **index** vengono passati ai metodi **read** e **write** della proprietà array.

L'esempio seguente mostra alcune dichiarazioni di proprietà semplici:

```
class PropertyExample {
  private:
    int Fx,Fy;
    float Fcells[100][100];

  protected:
    int  readX()          { return(Fx); }
    void writeX(int newFx) { Fx = newFx;  }

    double computeZ() {
          // Do some computation and return a floating point value...
```

```
            return(0.0);
        }


        float cellValue(int row, int col) { return(Fcells[row][col]); }

    public:
        __property int    X = { read=readX, write=writeX };
        __property int    Y = { read=Fy };
        __property double Z = { read=computeZ };
        __property float Cells[int row][int col] = { read=cellValue };
};
```

Questo esempio mostra diverse dichiarazioni di proprietà. La proprietà *X* ha accesso in lettura e scrittura, attraverso le funzioni membro *readX* e *writeX*. La proprietà *Y* corrisponde direttamente alla variabile membro *Fy* ed è a sola lettura. La proprietà *Z* è un valore a sola lettura che viene calcolato; non è memorizzato nella classe come un membro dati. Infine, la proprietà *Cells* dimostra una proprietà array con due indici. L'esempio successivo mostra come si acceda a queste proprietà nel codice:

```
PropertyExample myPropertyExample;
myPropertyExample.X = 42;           // Evaluates to: myPropertyExample.WriteX(42);
int    myVal1 = myPropertyExample.Y; // Evaluates to: myVal1 = myPropertyExample.Fy;
double myVal2 = myPropertyExample.Z; // Evaluates to: myVal2 = myPropertyExample.ComputeZ();
float  cellVal = myPropertyExample[3][7]; // Evaluates to:
                                          // cellVal = myPropertyExample.cellValue(3,7);
```

Le proprietà hanno molte altre varianti e funzionalità non illustrate in questo esempio. Gli utenti possono:

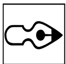
- associare lo stesso metodo di lettura o di scrittura a più di una proprietà, utilizzando l'attributo **index**
- specificare valori predefiniti
- memorizzare le proprietà in un file scheda
- estendere una proprietà definita in una classe base

Per maggiori informazioni sulle proprietà, consultare il Capitolo 47, "Creazione di proprietà".

__published

La parola chiave **__published** specifica che le proprietà di quella sezione vengono visualizzate nell'Object Inspector, se la classe si trova nella tavolozza Component. Solo le classi derivate da *TObject* possono avere sezioni **__published**.

Le regole di visibilità per i membri pubblicati sono identiche a quelle dei membri pubblici. L'unica differenza tra membri pubblicati e pubblici è che le informazioni di tipo runtime in stile (RTTI) di Object Pascal vengono generate per i membri dati e per le proprietà dichiarate in una sezione **__published**. RTTI consente ad un'applicazione di sottoporre a query in modo dinamico i membri dati, le funzioni membro e le proprietà di un tipo di classe altrimenti sconosciuto.



In una sezione **__published** non sono consentiti né costruttori né distruttori. Le proprietà, i membri dati Pascal intrinseci o derivati della VCL o della CLX, le funzioni membro e i closure sono ammessi solo in una sezione **__published**. I campi

definiti in una sezione **__published** devono essere di un tipo classe. Le proprietà definite in una sezione **__published** non possono essere proprietà array. Il tipo di una proprietà definito in una sezione **__published** deve essere un tipo ordinale, un tipo real, un tipo stringa, un tipo small **set**, un tipo classe, o un tipo puntatore a metodo.

Estensione della parola chiave __declspec

Alcuni argomenti per l'estensione della parola chiave **__declspec** forniscono il supporto del linguaggio per la VCL e la CLX. Questi argomenti sono elencati di seguito. Le macro per gli argomenti declspec e le loro combinazioni vengono definite in sysmac.h. Nella maggior parte dei casi non occorre specificarli. Quando è necessario aggiungerli, si devono usare le macro.

__declspec(delphiclass)

L'argomento **delphiclass** viene usato per le dichiarazioni di classi derivate da *TObject*. Queste classi vengono create con le seguenti compatibilità:

- RTTI compatibile con la VCL
- comportamento del costruttore/distruttore compatibile con la VCL
- gestione delle eccezioni compatibile con la VCL

Una classe compatibile con la VCL o la CLX ha le seguenti restrizioni:

- non è consentita alcuna classe base virtuale.
- non è consentita alcuna eredità multipla salvo nel caso descritto in "Eredità e interfacce" on page 13-2.
- deve essere allocata dinamicamente usando l'operatore globale new.
- deve avere un distruttore.
- i costruttori di copia e gli operatori di assegnazione non vengono generati dal compilatore per le classi derivate dalla VCL e dalla CLX.

Una dichiarazione di classe che viene tradotta da Object Pascal richiede questo modificatore se il compilatore ha bisogno di sapere se la classe è derivata da *TObject*.

__declspec(delphireturn)

In C++Builder, l'argomento **delphireturn** è usato solo internamente dalla VCL e dalla CLX. Viene usato per le dichiarazioni delle classi che sono state create in C++Builder per supportare i tipi di dati incorporati di Object Pascal e i costrutti del linguaggio, dato che essi non hanno un tipo C++ nativo. Questi includono *Currency*, *AnsiString*, *Variant*, *TDateTime* e *Set*. L'argomento **delphireturn** contrassegna le classi C++ per la gestione della compatibilità VCL e CLX in chiamate a funzioni come parametri e nella restituzione di valori. Questo modificatore è necessario quando si passa una struttura per valore ad una funzione tra Object Pascal e C++.

__declspec(delphirtti)

L'argomento **delphirtti** fa sì che il compilatore includa in una classe le informazioni di tipo in esecuzione quando tale classe viene compilata. Quando si utilizza questo modificatore, il compilatore genera informazioni di tipo in esecuzione per tutti i campi, metodi e proprietà che sono stati dichiarati in una sezione published. Nel caso delle interfacce, il compilatore genera le informazioni di tipo in esecuzione per tutti i metodi dell'interfaccia. Se una classe viene compilata con le informazioni di tipo in esecuzione, anche tutti i suoi discendenti includono le informazioni di tipo in esecuzione. Poiché la classe *TPersistent* viene compilato includendo le informazioni di tipo in esecuzione, ciò significa che non c'è alcuna necessità di utilizzare questo modificatore con le classe che vengono create e il cui antenato è *TPersistent*. Questo modificatore viene utilizzato principalmente per le interfacce in applicazioni che implementano o utilizzano Web Services.

__declspec(dynamic)

L'argomento **dynamic** viene usato per le dichiarazioni delle funzioni dinamiche. Queste funzioni sono simili alle funzioni virtuali, tranne per il fatto vengono memorizzate nelle vtables per gli oggetti che le definiscono, non nelle vtables dei discendenti. Se si chiama una funzione dinamica che non è definita nell'oggetto, la ricerca viene effettuata nelle vtables dei suoi antenati finché la funzione non viene trovata. Le funzioni dinamiche sono valide solo per le classi derivate da *TObject*.

__declspec(hidesbase)

L'argomento **hidesbase** preserva la semantica di programma di Object Pascal quando si esportano le funzioni virtuali di Object Pascal e si ridefiniscono per C++Builder. In Object Pascal le funzioni virtuali delle classi base possono apparire nella classe derivata come funzioni con lo stesso nome, destinate ad essere funzioni completamente nuove senza alcuna relazione esplicita con quelle precedenti.

I compilatori usano la macro HIDESBASE, definita in sysmac.h, per specificare che questi tipi di dichiarazione delle funzioni sono completamente diversi. Per esempio, se una classe base T1 dichiara una funzione virtuale, *func*, senza acquisire alcun argomento, e la sua classe derivata T2 ha dichiarato una funzione con lo stesso nome e signature, DCC32 -jphn produrrà un file HPP con il seguente prototipo:

```
virtual void T1::func(void);
HIDESBASE void T2::func(void);
```

Senza la dichiarazione HIDESBASE, la semantica di programma di C++ indica che la funzione virtuale *T1::func()* deve essere ridefinita dalla *T2::func()*.

__declspec(package)

L'argomento **package** indica che il codice che definisce la classe può essere compilato in un **package**. Questo modificatore viene generato automaticamente dal compilatore quando vengono creati i **package** nell'IDE. Per maggiori informazioni sui **package**, consultare il Capitolo 15, "Uso di package e di componenti".

__declspec(pascalimplementation)

L'argomento **pascalimplementation** indica che il codice che definisce la classe è stato implementato in Object Pascal. Questo modificatore appare in un file header di portabilità Object Pascal con estensione .hpp.

__declspec(uuid)

L'argomento **uuid** associa una classe con un identificativo globalmente univoco (GUID). Questi può essere utilizzato con qualsiasi classe, ma, di solito, viene utilizzato con classi che rappresentano interfacce di Object Pascal (o interfacce COM). È possibile ottenere il GUID di una classe che è stata dichiarata utilizzando questo modificatore chiamando la direttiva **__uuidof**.

Capitolo 14

Sviluppo di applicazioni multipiattaforma

È possibile utilizzare C++Builder per sviluppare applicazioni a 32 bit multipiattaforma eseguibili sia nel sistema operativo Windows sia nel sistema operativo Linux. Le applicazioni multipiattaforma utilizzano componenti CLX e non eseguono alcuna chiamata specifica alle API del sistema operativo. Per sviluppare un'applicazione multipiattaforma è possibile sia creare una nuova applicazione CLX sia modificare un'applicazione Windows esistente. Quindi la si deve compilare e distribuire sulla piattaforma su cui la si dovrà eseguire. Per l'ambiente Windows, utilizzare C++Builder. Per l'ambiente Linux, una soluzione C++ di Borland non è ancora disponibile, ma ci si può preparare per tempo sviluppando da ora le proprie applicazioni con C++Builder.

Questo capitolo descrive come modificare le applicazioni scritte in C++Builder in modo da poterle compilare in ambiente Linux e come scrivere del codice indipendente dalla piataforma e portabile fra i due ambienti. Questo capitolo include inoltre informazioni sulle differenze fra lo sviluppo di applicazioni nei due ambienti Windows e Linux.

Creazione di applicazioni multipiattaforma

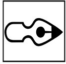
È possibile creare applicazioni multipiattaforma nello stesso modo in cui si creano le normali applicazioni in C++Builder. Se si vuole che l'applicazione sia perfettamente multipiattaforma è necessario utilizzare componenti visuali e non della CLX e non si dovrebbero utilizzare chiamate API specifiche del sistema operativo. (Per informazioni sulla scrittura di applicazioni multipiattaforma consultare "Scrittura di codice portabile" a pagina 14-17.)

Per creare un'applicazione multipiattaforma:

1 Nell'IDE, scegliere il comando File | New | CLX application.

La Component palette visualizzerà i componenti e le pagine che si possono utilizzare nelle applicazioni CLX.

2 Sviluppare l'applicazione all'interno dell'IDE.
3 Compilare e collaudare l'applicazione. Esaminare gli eventuali messaggi di errore per vedere dove è necessario apportare ulteriori modifiche.



Non appena sarà diponibile una soluzione C++Builder per Linux, sarà possibile compilare ed eseguire il collaudo in ambiente Linux.

Quando si eseguirà il porting dell'applicazione su Linux, sarà necessario impostare di nuovo le opzioni del progetto. Questo perché il file .dof che contiene le opzioni di progetto viene ricreato in Linux con una diversa estensione (con il set di opzioni predefinite).

I file scheda nella applicazioni multipiattaforma hanno l'estensione . xfm invece di.dfm. Questo per distingue le schede per multipiattaforma che utilizzano componenti CLX dalle schede che utilizzano componenti della VCL. Una scheda.xfm funziona su entrambe le piattaforme Windows e Linux mentre una scheda .dfm funziona solo in ambiente Windows.

Per informazioni sulla scrittura di applicazioni database o Internet indipendenti dalla piattaforma, consultare "Applicazioni database multipiattaforma" a pagina 14-21 and "Applicazioni Internet multipiattaforma" a pagina 14-30.

Porting di applicazioni Windows in Linux

Se si hanno applicazioni C++Builder scritte per l'ambiente Windows, è possibile convertirle per l'ambiente Linus. La facilità dell'operazione dipende dalla natura e dalla complessità dell'applicazione e da quante dipendenze esplicite a Windows essa include.

Le sezioni seguenti descrivono le principali differenze fra gli ambienti Windows e Linux e forniscono una serie di regole generali per iniziare immediatamente ad effettuare il porting di un'applicazione.

Tecniche di porting

Di seguito sono elencate le varie strade che è possibile intraprendere per trasportare un'applicazione da una piattaforma a una altra:

Tabella 14.1 Tecniche di porting

Tecnica	Descrizione
Porting specifico per una piattaforma	Ha come obiettivo un sistema operativo e le API sottostanti
Porting per più piattaforme	Ha come obiettivo un'API per più piattaforme
Emulazione Windows	Lascia inalterato il codice ed esegue il porting delle API utilizzate

Porting specifico per una piattaforma

I porting specifici per una certa piattaforma tendono a essere laboriosi, costosi e producono solo un singolo risultato mirato. Tali conversioni creano basi di codice differenti, il che rende il codice particolarmente difficile da gestire. Tuttavia, ogni porting è progettato per uno specifico sistema operativo e può sfruttare le funzionalità specifiche della piattaforma. Pertanto, l'applicazione di solito viene eseguita più rapidamente.

Porting per più piattaforme

Il porting per più piattaforme tende a far risparmiare tempo in quanto l'applicazione risultante ha come obiettivo più piattaforme. Nella realtà, la quantità di lavoro necessario per lo sviluppo di applicazioni multipiattaforma dipende fortemente dal codice esistente. Se il codice è stato sviluppato senza rispetto per l'indipendenza dalla piattaforma, ci si potrebbe imbattere in situazioni in cui la "logica" indipendente dalla piattaforma e l'"implementazione" dipendente dalla piattaforma risultano mischiate.

L'approccio multipiattaforma è l'approccio preferibile perché la logica di gestione viene espressa in termini indipendenti dalla piattaforma. Alcuni servizi vengono resi astratti e nascosti dietro un'interfaccia interna che ha lo stesso aspetto lo stesso su tutte le piattaforme, ma ha un'implementazione specifica per ognuna di esse. La libreria runtime di C++Builder ne è un tipico esempio. L'interfaccia è molto simile su entrambe le piattaforme, anche se limplementazione può essere notevolmente differente. Si dovrebbe separare le sezioni indipendenti dalla piattaforma, quindi implementare su di esse i servizi specifici. Alla fine, questo approccio risulta essere la soluzione meno dispendiosa, grazie ai costi di manutenzione ridotti derivanti dalla condivisione della maggior parte dei sorgenti di base e dal miglioramento dell'architettura dell'applicazione.

Porting in emulazione Windows

L'emulazione di Windows è il metodo più complesso e può risultare molto dispendioso, anche se l'applicazione Linux risultante sarà molto simile a un'applicazione Windows esistente. Questo approccio implica l'implementazione in Linux di funzionalità tipiche di Windows. Da un punto di vista architetturale, questa soluzione è molto difficile da gestire.

Nel caso si vogliano emulare le API di Windows, è possibile includere due sezioni distinte utilizzando le direttive **#ifdef** in modo da indicare le sezioni di codice dedicate specificatamente all'ambiente Windows o a quello Linux.

Porting dell'applicazione

Se si effettua il porting di un'applicazione che si vuole eseguire su entrambe le piattaforme Windows e Linux, sarà necessario modificare il codice oppure utilizzare direttive **#ifdef** per indicare le sezioni di codice dedicate specificatamente all'ambiente Windows o a quello Linux.

Per eseguire il porting di un'applicazione VCL in CLX, fare quanto segue:

1 Aprire in C++Builder il progetto che contiene l'applicazione che si desidera modificare.

2 Copiare i file .dfm in file .xfm aventi lo stesso nome (ad esempio, rinominare unit1.dfm in unit1.xfm). Rinominare (o includere in direttive **#ifdef**) il riferimento al file .dfm nel file header da `#pragma resource "*.dfm"` to `#pragma resource "*.xfm"`. (Il file .xfm funzionerà sia in applicazioni C++Builder sia in Linux.)

3 Modificare (o includere in direttive **#ifdef**) tutti i file header nel file sorgente in modo che facciano riferimento alle corrette unit nella CLX. (Per informazioni, consultare "Confronto fra le unit CLX e VCL" a pagina 14-9.)

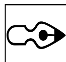
Ad esempio, modificare le seguenti istruzione #include nel file header di un'applicazione Windows:

```
#include <vcl.h>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
```

to the following for a CLX application:

```
#include <clx.h>
#include <QControls.hpp>
#include <QStdCtrls.hpp>
#include <QForms.hpp>
```

4 Salvare il progetto e riaprirlo. Ora la Component palette visualizzerà i componenti che si possono utilizzare nelle applicazioni CLX.



Nelle applicazioni CLX possono essere utilizzati solo alcuni componenti non visuali di Windows che però contengono funzionalità che possono essere utilizzate solo in applicazioni CLX di Windows. Se si prevede di compilare l'applicazione anche su Linux, non utilizzare componenti della VCL non visuali oppure utilizzare istruzioni condizionali **#ifdef** per contrassegnare queste sezioni di codice come utilizzabili solo per Windows. Non è possibile utilizzare nella stessa applicazione la parte visuale della VCL con VisualCLX.

5 Riscrivere tutto il codice che richiede dipendenze Windows, rendendolo il più possibile indipendente dalla piattaforma. A questo scopo utilizzare le routine e le costanti della libreria di runtime. (Per informazioni consultare "Scrittura di codice portabile" a pagina 14-17.)

6 Trovare le funzionalità equivalenti per quelle funzioni che sono diverse in Linux. Utilizzare **#ifdef** (con moderazione) per delimitare le informazioni specifiche per Windows. (Per informazioni consultare "Utilizzo delle direttive condizionali" a pagina 14-18.)

Ad esempio, nel file sorgente è possibile delimitare con direttive **#ifdef** il codice specifico per la piattaforma:

```
#ifdef WINDOWS //If using the C++Builder compiler, use __WIN32__
IniFile->LoadfromFile("c:\\x.txt");
#endif
```

```
#ifdef __linux__
IniFile->LoadfromFile("/home/name/x.txt");
#endif
```

7 Cercare in tutti i file di progetto i riferimenti ai nomi di percorso.

- Nei percorsi di Linux si utilizza come delimitatore una barra diritta / (ad esempio, /usr/lib) e i file possono essere situati in varie directory nel sistema Linux. Utilizzare la costante PathDelim (in SysUtils) per specificare il delimitatore di percorso appropriato per il sistema. Determinare l'ubicazione corretta per tutti i file in Linux.
- Modificare i riferimenti alle lettere di unità (ad esempio, C: \) e il codice che ricava le lettere associate alle unità ricercando il carattere due punti alla posizione 2 nella stringa. Utilizzare la costante DriveDelim (in SysUtils) per specificare l'ubicazione in modo appropriato per il sistema.
- Nei segmenti di codice in cui si specificano percorsi multipli, modificare il separatore di percorso da punto e virgola (;) a due punti (:). Utilizzare la costante PathSep (in SysUtils) per specificare il separatore di percorso appropriato per il sistema.
- Poiché in Linux i nomi dei file sono sensibili all'uso delle maiuscole e minuscole, assicurarsi che l'applicazione non modifichi i caratteri maiuscoli/minuscoli dei nomi dei file o non faccia affidamento su di essi.

8 Compilare, collaudare ed eseguire il debug dell'applicazione.

CLX e VCL

CLX utilizza la Component Library per Cross Platform (CLX) di Borland invece della Visual Component Library (VCL). All'interno della VCL, molti controlli consentono di accedere facilmente ai controlli Windows eseguendo chiamate nelle librerie API di Windows. Analogamente, la CLX consente l'accesso ai Qt widget (il nome deriva dalle parole window è gadget) nelle librerie condivise Qt. C++Builder include sia la CLX sia la VCL.

La CLX è molto simile alla VCL. La maggior parte dei componenti e delle proprietà hanno gli stessi nomi. Inoltre, la CLX, come anche la VCL, è disponibile in ambiente Windows (a seconda dell'edizione di C++Builder che si possiede).

I componenti CLX possono essere raggruppati come segue:

Tabella 14.2 Sezioni della CLX

Sezione	Descrizione
VisualCLX	Componenti GUI e grafici nativi multipiattaforma.
DataCLX	Componenti client per l'accesso ai dati. I componenti in questa sezione sono un sottoinsieme dei componenti locali, client/server e n-tier basati sui dataset client. Il codice è lo stesso in Linux e in Windows.

Tabella 14.2 Sezioni della CLX

Sezione	Descrizione
NetCLX	Componenti Internet tra cui Apache DSO e CGI WebBroker. Questi sono identici in Linux e in Windows.
BaseCLX	Libreria di runtime aggiornata e comprensiva di unit Classes. Il codice è lo stesso in Linux e in Windows.

I widget in VisualCLX sostituiscono i controlli Windows. Ad esempio, *TWidgetControl* nella CLX sostituisce *TWidgetControl* nella VCL. Altri componenti della VCL (come TScrollingWincontrol) hanno nomi corrispondenti nella CLX (come *TScrollingWidget*). Tuttavia, non è necessario modificare ad esempio tutte le ricorrenze di *TWinControl* in *TWidgetControl*. Dichiarazioni di tipo, come ad esempio:

```
TWinControl = TWidgetControl;
```

appaiono nel file unit QControls per semplificare la condivisione del codice sorgente. *TWidgetControl* e tutti i suoi discendenti hanno una proprietà *Handle* che referenzia l'oggetto Qt e una proprietà *Hooks* che referenzia l'oggetto hook object che gestisce il meccanismo degli eventi.

Nella CLX i nomi delle unit e l'ubicazione di alcune classi sono differenti. Sarà necessario modificare il file header incluso nel file sorgente per eliminare i riferimenti alle unit che non esistono nella CLX e modificare i nomi in quelli delle unit della CLX.

Differenze nella CLX

Benché quasi tutta la CLX sia implementata in modo da essere compatibile con la VCL, alcune funzioni sono implementate diversamente. Questa sezione fornisce una panoramica su alcune differenze esistenti fra le implementazioni della CLX e della VCL che occorre tener presente durante la scrittura di applicazioni multipiattaforma.

Aspetto

L'ambiente visuale in Linux è abbastanza differente rispetto a Windows. L'aspetto delle finestre di dialogo può essere diverso a seconda del Windows manager utilizzato, come ad esempio KDE o Gnome

Stili

Oltre alle proprietà *OwnerDraw* è possibile utilizzare “ stili” a livello di applicazione. È possibile utilizzare la proprietà *TApplication::Style* per specificare l'aspetto degli elementi grafici di un'applicazione. Utilizzando gli stili, un widget o un'applicazione può assumere un aspetto completamente differente. In Linux è sempre possibile utilizzare uno stile owner draw ma si consiglia l'utilizzo degli stili.

Variant

Tutto il codice relativo a variant e safe array che era inserito nella unit System è incluso in due nuove unit:

- Variants
- VarUtils

Il codice dipendente dal sistema operativo è ora isolato nella unit VarUtils, che contiene anche versioni generiche di tutto ciò che le è necessario. Se si sta convertendo un'applicazione VCL che includeva chiamate Windows in un'applicazione CLX, è necessario sostituire queste chiamate con chiamate nella unit VarUtils.

Se si desidera utilizzare dei variant, è necessario includere la unit Variants al file header del file sorgente.

VarIsEmpty esegue un semplice controllo su *varEmpty* per vedere se un variant è vuoto mentre su Linux è possibile utilizzare la funzione *VarIsClear* per verificare se il valore del variant è indefinito.

File di registro

Linux non utilizza un file di registro per memorizzare le informazioni di configurazione. Invece, al posto del file di registro, si utilizzano file di configurazione di testo e variabili di ambiente. I file di configurazione di sistema su Linux sono situati spesso in /etc, ad esempio, in /etc/host. Altri profili utente sono situati in file nascosti (preceduti da un punto), come .bashrc, che contiene impostazioni della shell bash, oppure .XDefaults, utilizzato per memorizzare le impostazioni predefinite dei programmi X.

Il codice dipendente dal file di registro può essere modificato in modo da utilizzare al suo posto un file di testo di configurazione locale. Le impostazioni modificabili dall'utente possono essere salvate nella directory personale in modo che l'utente abbia i diritti di scrittura. Le opzioni di configurazione che è necessario impostare dalla radice dovrebbero andare in /etc.Un modo per gestire una precedente dipendenza dal file di registro è quello di scrivere una unit che includa tutte le funzioni relative al file di registro, dirigendo però tutto l'output su un file di configurazione locale.

Se si vogliono collocare delle informazioni in una posizione globale in ambiente Linux, è possibile registrare un file di configurazione globale nella directoryy /etc o nella directory home dell'tente come file nascosto. In questo modo tutte le applicazioni potranno accedere allo stesso file di configurazione. Tuttavia, è necessario accertarsi che le autorizzazioni sui file e i diritti di accesso siano configurati correttamente.

Anche nelle applicazioni multipiattaforma è possibile utilizzare i file .ini. Tuttavia, nella CLX, è necessario utilizzare *TMemIniFile* invece di *TRegIniFile.*

Altre differenze

L'implementazione di CLX presenta anche altre differenze rispetto alla VCL che influiscono sulla modalità di funzionamento dei componenti. Questa sezione descrive alcune di queste differenze.

- È possibile selezionare un componente CLX dalla Component palette e fare indifferentemente clic col pulsante destro o sinistro per aggiungerlo a una scheda. Nel caso di componenti VCL, è possibile solo fare clic col pulsante sinistro.

- Il controllo *TButton* della CLX ha una proprietà *ToggleButton* che l'equivalente controllo della VCL non possiede.
- In CLX, *TColorDialog* non ha una proprietà *TColorDialog::Options* da impostare. Quindi, è impossibile personalizzare l'aspetto e le funzionalità della finestra di dialogo dei colori. Inoltre, a seconda del window manager utilizzato in Linux, *TColorDialog* non sempre è modale o con dimensioni fisse. In ambiente Windows, *TColorDialog* è sempre modale e non dimensionabile.
- In fase di esecuzione, le caselle combinate della CLX funzionano in modo diverso rispetto a quelle della VCL. Nella CLX (ma non nella VCL), è possibile aggiungere un elemento a un elenco a discesa immettendo del testo nel campo di editing di una casella combinata e premendo il tasto Invio. È possibile disattivare questa funzione impostando *InsertMode* a *ciNone*. È anche possibile aggiungere elementi vuoti (nessuna stringa) alla lista della casella combinata. Inoltre, se si tiene premuto il tasto freccia giù, non ci si ferma sull'ultimo elemento dell'elenco della casella combinata. Si torna nuovamente all'inizio.
- *TCustomEdit* non implementa *Undo*, *ClearUndo* o *CanUndo*. Pertanto non esiste alcun modo per annullare da programma le modifiche. In fase di esecuzione, gli utenti dell'applicazione potranno comunque annullare le modifiche apportate a una casella di testo (*TEdit*) facendo clic destro sulla casella di modifica e scegliendo il comando *Undo*.
- I valori dei tasti usati negli eventi possono essere differenti fra la VCL e la CLX. Ad esempio, il tasto Invio ha un valore pari a 13 nella VCL e un valore pari a 4100 nella CLX. Se nelle applicazioni CLX il valore dei tasti è stato inserito nel codice, sarà necessario modificare tali valori quando si eseguirà il porting da Windows a Linux o viceversa.

Esistono anche altre differenze. Per i dettagli su tutti gli oggetti della CLX, fare riferimento alla documentazione in linea relativa alla CLX, oppure, nelle versioni di C++Builder che includono il codice sorgente, è possibile consultare direttamente il codice che è situato nella directory {directory di installazione}\C++Builder6\Source\CLX.

Cosa manca nella CLX

Se si utilizza la CLX invece della VCL, si potrà notare che molti oggetti sono identici. Tuttavia, gli oggetti potrebbero non avere tutte le funzionalità (come proprietà, metodi o eventi). Nella CLX mancano le seguenti funzionalità generali:

- Proprietà bidirezionali (*BidiMode*) per input/output di testo da destra a sinistra.
- Proprietà bevel generiche sui controlli comuni (si noti che alcuni oggetti hanno ancora proprietà bevel).
- Proprietà e metodi per l'aggancio.
- Funzionalità per la compatibilità con versioni precedenti come i componenti sulle pagina Win3.1 e *Ctl3D.*
- *DragCursor* e *DragKind* (anche se le funzioni drag-and-drop sono incluse).

Funzioni di cui non è possibile fare il porting

Alcune caratteristiche specifiche di Windows supportate in C++Builder non possono essere portate direttamente in ambienti Linux. Ad esempio, funzionalità quali COM, ActiveX, OLE, BDE e ADO sono dipendenti dalla tecnologia Windows e non sono disponibili per Linux. La tabella seguente elenca le funzionalità che differiscono sulle due piattaforme e, nel caso sia disponibile, elenca l'equivalente funzionalità Linux o della CLX.

Tabella 14.3 Funzionalità modificate o differenti

Funzionalità Windows/VCL	Funzionalità Linux/CLX
Componenti ADO	Normali componenti database
Automation Server	Non disponibili
BDE	dbExpress e normali componenti database
Componenti COM (inclusi ActiveX)	Non disponibili
DataSnap	Non disponibili
FastNet	Non disponibili
Componenti precedenti (come gli elementi sulla pagina Win 3.1 della Component palette)	Non disponibili
Messaging Application Programming Interface (MAPI) include una libreria standard di funzioni Windows per l'invio di messaggi.	SMTP e POP3 permettono di inviare, ricevere e salvare messaggi di posta elettronica
Quick Reports	Non disponibili
Web Services (SOAP)	Non disponibili
WebSnap	Non disponibili
Chiamate alle API di Windows	Metodi CLX, chiamate QT, chiamate alla libc o chiamate ad altre librerie di sistema
Invio messaggi Windows	Eventi Qt
Winsock	BSD sockets

L'equivalente Linux delle DLL di Windows sono le librerie di oggetti condivisi (file .so), che contengono codice indipendente dalla posizione (PIC). Quindi, i riferimenti di memoria globali e le chiamate a funzioni esterne sono tutti relativi al registro EBX, che deve essere salvaguardato durante le chiamate.

È necessario preoccuparsi dei riferimenti di memoria globali e delle chiamate a funzioni esterne solo nel caso si utilizzi l'assembler - C++Builder è in grado di generare il codice corretto. (Per informazioni, vedere "Inclusione di codice assembler in linea" a pagina 14-20.)

Confronto fra le unit CLX e VCL

Tutti gli oggetti nella VCL o nelle CLX sono definiti in file header. Ad esempio, è possibile trovare l'implementazione di *TObject* nella unit System e la unit Classes

definisce la classe di base *TComponent*. Quando si rilascia un oggetto su una scheda o si utilizza un oggetto all'interno dell'applicazione, il nome della unit viene aggiunto al file header incluso nel file sorgente, che dice al compilatore quali sono le unit di cui eseguire il link nel progetto.

Questa sezione presenta tre tabelle che elencano le unit equivalenti fra VCL e CLX units, le unit specifiche per la CLX e le unit specifiche per la VCL.

La seguente tabella elenca la unit nella VCL e quella equivalente nella CLX. Non sono riportate le unit identiche nella VCL e nella CLX, oppure le unit fornite da produttori terzi.

Tabella 14.4 Unit nella VCL ed equivalenti nella CLX

Unit della VCL	Unit della CLX
ActnList	QActnList
Buttons	QButtons
CheckLst	QCheckLst
Clipbrd	QClipbrd
ComCtrls	QComCtrls
Consts	Consts, QConsts, and RTLConsts
Controls	QControls
DBActns	QDBActns
DBCtrls	QDBCtrls
DBGrids	QDBGrids
Dialogs	QDialogs
ExtCtrls	QExtCtrls
Forms	QForms
Graphics	QGraphics
Grids	QGrids
ImgList	QImgList
Mask	QMask
Menus	QMenus
Printers	QPrinters
Search	QSearch
StdActns	QStdActns
StdCtrls	QStdCtrls
Types	Types and QTypes
VclEditors	ClxEditors

Le seguenti unit sono nella CLX ma non nella VCL:

Tabella 14.5 Unit solo nella CLX

Unit	Descrizione
DirSel	Selezione di directory

Tabella 14.5 Unit solo nella CLX

Unit	Descrizione
QStyle	Aspetto della GUI
Qt	Interfaccia per la libreria Qt

Le seguenti unit VCL per Windows non sono incluse nella CLX principalmente perché sono relative a funzionalità specifiche di Windows, come ADO, COM e BDE, che non sono disponibili in Linux.

Tabella 14.6 Unit solo nella VCL

Unit	Motivo dell'esclusione
ADOConst	Nessuna funzionalità ADO
ADODB	Nessuna funzionalità ADO
AppEvnts	Nessun oggetto TApplicationEvent
AxCtrls	Nessuna funzionalità COM
BdeConst	Nessuna funzionalità BDE
Calendar	Non supportata
Chart	Non supportata
CmAdmCtl	Nessuna funzionalità COM
ColorGrd	Non supportata
ComStrs	Nessuna funzionalità COM
ConvUtils	Non disponibili
CorbaCon	Nessuna funzionalità Corba
CorbaStd	Nessuna funzionalità Corba
CorbaVCL	Nessuna funzionalità Corba
CtlPanel	Nessun supporto per Windows Control Panel
CustomizeDlg	Non supportata
DataBkr	Non supportata
DBCGrids	Nessuna funzionalità BDE
DBExcept	Nessuna funzionalità BDE
DBInpReq	Nessuna funzionalità BDE
DBLookup	Obsoleta
DbOleCtl	Nessuna funzionalità COM
DBPWDlg	Nessuna funzionalità BDE
DBTables	Nessuna funzionalità BDE
DdeMan	Nessuna funzionalità DDE
DRTable	Nessuna funzionalità BDE
ExtActns	Non supportata al momento
ExtDlgs	Nessuna funzionalità per finestre di dialogo con picture
FileCtrl	Obsoleta

Tabella 14.6 Unit solo nella VCL

Unit	Motivo dell'esclusione
ListActns	Non supportata
MConnect	Nessuna funzionalità COM
Messages	Nessuna gestione messaggi di Windows
MidasCon	Obsoleta
MPlayer	Non esiste media player di Windows
Mtsobj	Nessuna funzionalità COM
MtsRdm	Nessuna funzionalità COM
Mtx	Nessuna funzionalità COM
mxConsts	Nessuna funzionalità COM
ObjBrkr	Non supportata
OleConstMay	Nessuna funzionalità COM
OleCtnrs	Nessuna funzionalità COM
OleCtrls	Nessuna funzionalità COM
OLEDB	Nessuna funzionalità ADO
OleServer	Nessuna funzionalità COM
Outline	Obsoleta
Registry	Nessuna funzionalità per il registry di Windows
ScktCnst	Sostituito da Sockets
ScktComp	Sostituito da Sockets
SConnect	Protocolli di connessione non supportati
SHDocVw_ocx	Nessuna funzionalità ActiveX
StdConvs	Non supportata
SvcMgr	Nessuna funzionalità per i servizi NT di Windows
TabNotbk	Obsoleta
Tabs	Obsoleta
ToolWin	Nessuna funzionalità di aggancio
ValEdit	Non supportata al momento
VarCmplx	Non supportata al momento
VarConv	Non supportata al momento
VCLCom	Nessuna funzionalità COM
WebConst	Nessuna costante Windows
Windows	Nessuna chiamata alle API di Windows

Differenze nei costruttori di oggetti CLX

Quando si crea un oggetto CLX, o implicitamente, collocando quell'oggetto sulla scheda, o esplicitamente nel codice, utilizzando il costruttore dell'oggetto, viene creata anche un'istanza del widget associato sottostante. L'oggetto CLX possiede questa istanza del widget. Quando l'oggetto della CLX viene cancellato viene cancellato anche il widget sottostante. Questo è lo stesso tipo di funzionalità che è possibile trovare nella VCL nelle applicazioni Windows.

Quando nel codice si crea esplicitamente un oggetto CLX mediante una chiamata alla libreria delle interfacce di QT come QWidget_Create(), si sta creando un'istanza di un widget QT che non è posseduto da un oggetto CLX. In questo modo si passa all'oggetto CLX l'istanza di un widget Qt esistente da utilizzare durante la sua costruzione. Questo oggetto CLX non possiede il widget Qt che gli viene passato. Quindi, dopo avere creato l'oggetto in questo modo, viene distrutto solo l'oggetto CLX e non l'istanza del widget Qt sottostante. Questo comportamento è diverso rispetto a quello della VCL.

Alcuni oggetti grafici della CLX, come *TBrush* e *TPen*, permettono di assumere la proprietà del widget sottostante utilizzando il metodo *OwnHandle*. Dopo avere chiamato *OwnHandle*, se si cancella l'oggetto CLX, viene distrutto anche il widget sottostante.

Alcune assegnazioni di proprietà nella CLX sono state spostate dal costruttore a *InitWidget*. Ciò consente di ritardare la costruzione dell'oggetto Qt finché non sarà veramente necessario. Ad esempio, si supponga di avere una proprietà di nome *Color*. In *SetColor*, è possibile interrogare *HandleAllocated* per controllare se si dispone di un handle Qt. Se lo handle è allocato, è possibile effettuare la chiamata corretta a Qt per impostare il colore. Nel caso non lo sia, è possibile memorizzare il valore in una variabile di campo privata e impostare la proprietà in *InitWidget*.

Per maggiori informazioni sulla costruzione di oggetti, vedere "Costruzione di oggetti per le classi VCL\CLX di C++Builder" a pagina 13-8.

Gestione degli eventi di sistema e di widget

Gli eventi di sistema e di widget, che sono la preoccupazione principale quando si scrivono componenti, sono gestiti in modo diffferente dalla VCL e dalla CLX. La differenza più importante è che i controlli CLX non rispondono direttamente ai messaggi di Windows, anche quando sono in esecuzione in ambiente Windows (vedere il Capitolo 51, "Gestione dei messaggi e delle notifiche di sistema"). Invece, essi rispondono alle notifiche provenienti dallo strato del widget sottostante. Poiché le notifiche utilizzano un sistema diverso, l'ordine e la sincronizzazione degli eventi possono essere diversi talvolta tra oggetti CLX e VCL corrispondenti. Tale differenza è riscontrabile anche quando l'applicazione CLX viene eseguita in ambiente Windows piuttosto che in Linux. Se si esegue il porting di un'applicazione VCL a CLX, per gestire queste differenze potrebbe essere necessario modificare il modo in cui i gestori di evento reagiscono agli eeventi.

Per informazioni sulla scrittura di componenti che rispondono a eventi di sistema e di widget (diversi da quelli previsti negli eventi resi pubblici dei componenti CLX), vedere "Risposta alle notifiche di sistema utilizzando la CLX" a pagina 51-11.

Condivisione di file sorgente tra Windows e Linux

Se si desidera che l'applicazione sia eseguita sia in Windows che in Linux, è possibile condividere i file sorgente rendendoli accessibili a entrambi i sistemi operativi. È possibile fare ciò in molti modi, come, ad esempio, mettendo i file sorgente su un server accessibile a entrambi i computer oppure utilizzando Samba sulla macchina Linux per consentire l'accesso ai file mediante la condivisione di file in rete Microsoft sia per Linux che per Windows. È possibile decidere di tenere i sorgenti su Linux e creare un'unità condivisa su Linux. Oppure è possibile tenere i sorgenti su Windows e creare una condivisione in Windows affinché la macchina Linux possa accedervi.

È possibile continuare a sviluppare e compilare il file su C++Builder utilizzando oggetti supportati sia dalla VCL sia dalla CLX. Una volta terminato, è possibile compilare su Windows. Appena sarà disponibile una soluzione C++ per Linux sarà possibile eseguire la compilazione in Linux.

Se si crea una nuova applicazione CLX, in C++Builder viene creato un file .xfm invece di un file .dfm.

Differenze di ambiente fra Windows e Linux

Attualmente, il termine multipiattaforma sta ad indicare un'applicazione che può essere eseguita, praticamente senza modifiche, su entrambi i sistemi operativi Windows e Linux. La seguente tabella elenca alcune differenze esistenti fra gli ambienti operativi Linux e Windows.

Tabella 14.7 Differenze fra gli ambienti operativi Linux e Windows

Differenze	Descrizione
Sensibilità alle maiuscole/minuscole nei nomi di file	In Linux, i nomi dei file risentono delle differenze fra maiuscole e minuscole. Il file Test.txt è un file differente rispetto al file test.txt. In Linux è necessario prestare grande attenzione all'uso delle maiuscole/minuscole nei nomi dei file.
Caratteri di fine riga	In Windows, le righe di testo terminano con CR/LF (cioè, ASCII 13 + ASCII 10), mentre in Linux viene usato solo LF. Benché l'editor di codice sia in grado di gestire la differenza, è bene ricordarsene quando si importa del codice in Windows.
Carattere di fine file	In MS-DOS e in Windows, il carattere con valore #26 (Ctrl-Z) viene trattato come la fine del file di testo, anche se, dopo quel carattere, nel file vi sono dati. Linux utilizza Ctrl+D come carattere di fine file.
Batch file/ procedure di shell	L'equivalente Linux dei file .bat sono le procedure di shell. Una procedura è un file di testo contenente istruzioni, salvate e rese eseguibile con il comando, `chmod +x <scriptfile>`. Il linguaggio procedurale dipende dalla shell utilizzata su Linux. Bash è quello usato più di frequente.

Tabella 14.7 Differenze fra gli ambienti operativi Linux e Windows

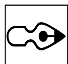
Differenze	Descrizione
Conferma dei comandi	In MS-DOS o in Windows, se si prova a cancellare un file o una cartella, apparirà una richiesta di conferma (un messaggio del tipo "Si conferma l'operazione?"). Di solito, Linux non fa alcuna domanda; eseguirà l'ordine o il comando. In questo modo è molto facile distruggere casualmente un file o l'intero file system. In Linux non c'è alcun modo per annullare una cancellazione a meno che non sia stata fatta una copia di riserva del file su altro supporto.
Risultato dei comandi	In Linux, se un comando va a buon fine, viene rivisualizzato il prompt dei comandi senza alcun messaggio di stato.
Parametri dei comandi	Linux utilizza un trattino (-) per indicare un parametro di comando o un trattino doppio (--) per più caratteri di opzioni mentre il DOS utilizza una barra (/) o un trattino (-).
File di configurazione	In Windows, la configurazione viene fatta nel file di registro o in file come autoexec.bat. In Linux, i file di configurazione vengono creati come file nascosti nella directory home dell'utente. I file di configurazione presenti nella directory /etc di solito non sono nascosti. Linux utilizza anche variabili di ambiente come LD_LIBRARY_PATH (percorso di ricerca per le librerie). Altre variabili di ambiente importanti: HOME La directory iniziale dell'utente (/home/sam) TERM Tipo di terminale (xterm, vt100, console) SHELL percorso per la shell (/bin/bash) USER Il nome di login dell'utente (sfuller) PATH Elenco di ricerca dei programmi Queste variabili sono specificate nella shell o nei file come ad esempio .bashrc.
DLL	Su Linux, si utilizzano file di oggetti condivisi (.so). I corrispondenti Windows sono le librerie a collegamento dinamico (DLL).
Lettere di unità	Linux non ha lettere di unità. Un esempio di percorso Linux è /lib/security. Fare riferimento a DriveDelim nella libreria di runtime.
Eccezioni	Le eccezioni di sistema operativo in Linux vengono chiamate segnali.
File eseguibili	In Linux, i file eseguibili non richiedono alcuna estensione. In Windows, i file eseguibili hanno l'estensione exe.
Estensioni dei nomi di file	Linux non utilizza estensioni dei nomi di file per identificare i tipi di file o per associare i file alle applicazioni.
Permessi sui file	In Linux, ai file (e alle directory) vengono assegnati i diritti di lettura, scrittura ed esecuzione al proprietario di file, al gruppo e ad altri. Ad esempio, `-rwxr-xr-x` significa, da sinistra a destra: - è il tipo di file (- = file comune, d = directory, l = collegamento); rwx sono i permessi concessi al proprietario del file (lettura, scrittura, esecuzione); r-x sono i permessi concessi al gruppo del proprietario del file (lettura, esecuzione); e r-x sono i permessi concessi a tutti gli altri utenti (lettura, esecuzione). L'utente root (superuser) può ridefinire questi permessi. È necessario accertarsi che l'applicazione venga eseguita dall'utente corretto e abbia i corretti accessi per i file necessari.

Tabella 14.7 Differenze fra gli ambienti operativi Linux e Windows

Differenze	Descrizione
Programma di utilità Make	Il programma di utilità Make di Borland non è disponibile sulla piattaforma Linux. Al suo posto è possibile utilizzare il programma di utilità Make GNU proprio di Linux.
Multitasking	Linux supporta completamente il multitasking. È possibile eseguire contemporaneamente svariati programmi (in Linux, vengono chiamati processi). È possibile avviare processi in background (utilizzando il simbolo & dopo il comando) e continuare a operare tranquillamente. Linux consente anche di avere diverse sessioni.
Percorsi	Linux utilizza una barra (/) mentre il DOS utilizza una barra retroversa (\). Per specificare il carattere appropriato per la piattaforma è possibile utilizzare la costante PathDelim. Fare riferimento a PathDelim nella libreria di runtime.
Percorsi di ricerca	Per l'esecuzione dei programmi, Windows controlla sempre, come prima cosa, la directory corrente, quindi guarda nei percorsi specificati nella variabile di ambiente PATH. Linux non guarda mai nella directory corrente ma cerca soltanto nelle directory elencate in PATH. Per eseguire un programma nella directory corrente, di solito è necessario immettere il carattere ./ prima del nome del programma. È possibile anche modificare la variabile PATH in modo da includere ./ come primo percorso di ricerca.
Separatore del percorso di ricerca	Windows utilizza il punto e virgola come separatore dei percorsi di ricerca. Linux utilizza il carattere due punti. Fare riferimento a PathDelim nella libreria di runtime.
Collegamenti simbolici	Su Linux, un collegamento simbolico è un file speciale che punta a un altro file su disco. Se si collocano i collegamenti simbolici nella directory globale bin che punta ai file principali dell'applicazione non sarà necessario modificare il percorso di ricerca di sistema. Un collegamento simbolico viene creato con il comando ln (link). Windows ha i collegamenti del desktop. Di solito, per rendere un programma disponibile dalla riga comandi, i programmi di installazione Windows modificano il percorso di ricerca di sistema.

Struttura delle directory su Linux

In Linux le directory sono diverse. Qualsiasi file o dispositivo può essere montato ovunque nel file system.



I nomi dei percorsi in Linux utilizzano le barre diritte mentre i nomi dei percorsi in Windows utilizzano le barre retroverse. La barra iniziale indica la directory radice.

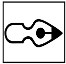
Di seguito sono elencate alcune directory comunemente utilizzate in Linux.

Tabella 14.8 Directory Linux comuni

Directory	Contentuto
/	La directory radice o superiore dell'intero file system Linux
/root	La radice del file system; la home directory del Superuser
/bin	Comandi, programmi di utilità

Tabella 14.8 Directory Linux comuni (continua)

Directory	Contentuto
/sbin	Programmi di utilità di sistema
/dev	Dispositivi visualizzati come file
/lib	Librerie
/home/username	File di proprietà dall'utente in cui username è il nome del login dell'utente.
/opt	Accessori
/boot	Kernel che viene chiamato all'avvio del sistema
/etc	File di configurazione
/usr	Applicazioni, programmi. Di solito include directory come /usr/spool, / usr/man, /usr/include, / usr/local
/mnt	Altri supporti montati sul sistema come un CD o un'unità per dischetti
/var	File di registrazione, messaggi, file di spool
/proc	Statistiche del file system virtuale e del sistema di reporting
/tmp	File temporanei



Le varie distribuzioni di Linux collocano talvolta i file in posizioni differenti. Un programma di utilità potrebbe essere collocato in /bin da una distribuzione Red Hat e collocato invece in /usr/local/bin da una distribuzione Debian.

Consultare www.pathname.com per ulteriori dettagli sull'organizzazione del file system gerarchico dell'ambiente UNIX/Linux e leggere il documento *Filesystem Hierarchy Standard*.

Scrittura di codice portabile

Se si stanno scrivendo applicazioni multipiattaforma destinate a funzionare in Windows e in Linux, è possibile scrivere del codice che venga compilato in base a diverse condizioni. Utilizzando la compilazione condizionale, è possibile conservare la codifica Windows consentendo al contempo le differenze dovute al sistema operativo Linux.

Per creare applicazioni che siano facilmente portabili tra Windows e Linux, è bene ricordare di:

- ridurre o isolare le chiamate alle API specifiche della piattaforma (Win32 o Linux); utilizzare invece i metodi della CLX oppure chiamate alla libreria QT.
- eliminare dall'applicazione i costrutti relativi alla gestione dei messaggi Windows (PostMessage, SendMessage). In CLX, chiamare invece i metodi *QApplication_postEvent* e *QApplication_sendEvent*. Per informazioni sulla scrittura di componenti che rispondono a eventi di sistema e di widget, consultare "Risposta alle notifiche di sistema utilizzando la CLX" a pagina 51-11.
- utilizzare *TMemIniFile* invece di *TRegIniFile*.
- osservare e proteggere l'utilizzo di maiuscole/minuscole nei nomi di file e di directory.

- Fare il porting di tutto il codice TASM prodotto dall'assembler esterno. L'assembler GNU, "as," non supporta la sintassi TASM. (Vedere "Inclusione di codice assembler in linea" a pagina 14-20.)

Provare a scrivere il codice utilizzando le routine di libreria runtime indipendenti dalla piattaforma e le costanti incluse in System, SysUtils e in altre unit della libreria runtime. Ad esempio, utilizzare la costante PathDelim per isolare quelle parti di codice che, a seconda della piattaforma, utilizzano la barra / al posto della barra retroversa.

Un altro esempio riguarda l'utilizzo di caratteri multibyte su entrambe le piattaforme. Il codice Windows normalmente si aspetta solo due byte per ogni carattere multibyte. In Linux, la codifica dei caratteri multibyte può richiedere molti più byte per carattere (fino a sei byte per UTF-8). Entrambe le piattaforme possono essere gestite utilizzando la funzione StrNextChar in SysUtils. Del codice Windows come quello che segue

```
while (*p != 0)
{
   if (LeadBytes.Contains(*p))
      p++;
   p++;
}
```

può essere sostituito con del codice indipendente dalla piattaforma simile al seguente:

```
while (*p != 0)
{
   if (LeadBytes.Contains(*p))
     p = StrNextchar(p);
   else
     p++;
}
```

Questo esempio è portabile su più piattaforme, ma tuttavia non consente di evitare ancora il costo, in termini di prestazioni, a una chiamata di procedura per ambienti non-multibyte.

Se l'utilizzo di funzioni di libreria di esecuzione non è una soluzione adottabile, provare a isolare il codice specifico per la piattaforma presente nella routine in un pezzo o in una subroutine. Provare a limitare il numero di blocchi **#ifdef** per conservare la leggibilità e la portabilità del codice sorgente. Il simbolo condizionale WIN32 non è definito in Linux. Il simbolo condizionale LINUX è definito, e indica che il codice sorgente viene compilato per la piattaforma Linux.

Utilizzo delle direttive condizionali

L'uso di direttive di compilazione **#ifdef** è un metodo accettabile per condizionare il codice in funzione delle piattaforme Windows e Linux. Tuttavia, poiché le direttive **#ifdef** rendono il codice sorgente più difficile da capire e da mantenere, è necessario capire quando è accettabile utilizzare le direttive di **#ifdef**. Se si presentasse l'occasione di utilizzare la direttiva **#ifdef**, le domande che ci si dovrebbero porre

sono “Perché questo codice richiede un **#ifdef**?” e “Può essere scritto senza un **#ifdef**?”

Per l’utilizzo di **#ifdef** all’interno di applicazioni multipiattaforma, seguire queste regole generali:

- Provare a non utilizzare **#ifdef** a meno che non sia assolutamente necessario. Le direttive **#ifdef** in un file sorgente vengono valutate solo al momento della compilazione del codice sorgente. C/C++ necessitano dei sorgenti della unit (file header) per compilare un progetto. La rigenerazione completa di tutto il codice sorgente è un evento insolito per la maggior parte dei progetti C++Builder.
- Non utilizzare **#ifdef** nei file package (.bpk). Limitare il loro utilizzo ai file sorgente. Gli scrittori di componenti devono creare due package distinti di progettazione quando sviluppano per più piattaforme, e non un solo package utilizzando **#ifdef**.
- In generale, utilizzare **#ifdef** WINDOWS per verificare tutte le piattaforme Windows, inclusa WIN32. Riservare l’utilizzo di **#ifdef** WIN32 per effettuare una distinzione fra piattaforme Windows specifiche, ad esempio tra Windows 32 bit e 64 bit. Non limitare il codice alla piattaforma WIN32 a meno che non si sappia per certo che non verrà utilizzato in WIN64.
- Evitare test di negazione come **#ifndef** a meno che non sia assolutamente necessario. **#ifndef __linux__** non è equivalente a **#ifdef** WINDOWS.
- Evitare combinazioni **#ifndef/#else**. Utilizzare invece un test positivo (**#ifdef**) per migliorare la leggibilità.
- Evitare clausole **#else** in **#ifdef** sensibili alla piattaforma. Usare blocchi **#ifdef** separati per Linux e codice specifico per Windows al posto di **#ifdef** __linux__/ **#else** oppure di **#ifdef** WINDOWS/**#else**.

  Ad esempio, del vecchio codice potrebbe contenere

  ```
  #ifdef WIN32
     (32-bit Windows code)
  #else
     (16-bit Windows code)   //!! By mistake, Linux could fall into this code.
  #endif
  ```

  Per qualsiasi codice non portatile in **#ifdef**, è preferibile che il codice sorgente non venga compilato piuttosto che la piattaforma utilizzi erroneamente una clausola **#else** e generi misteriosamente un errore in esecuzione. Gli errori di compilazione sono più facili da individuare dei malfunzionamenti in esecuzione.
- Utilizzare la sintassi **#if** per test complicati. Sostituire **#ifdef** annidati con un’espressione booleana in una direttiva **#if**. Si deve terminare la direttiva **#if** usando **#endif**. Ciò consente di collocare espressioni **#if** all’interno di direttive **#ifdef** in modo da nascondere la nuova sintassi **#if** ai compilatori precedenti.

Tutte le direttive condizionali sono documentate nella Guida in linea. Per ulteriori informazioni, consultare anche l’argomento “Conditional Compilation” nella Guida in linea.

Emissione di messaggi

La direttiva al compilatore **#pragma** consente al codice sorgente di emettere suggerimenti e avvertimenti ed errori, esattamente come il compilatore. Usare la direttiva **#pragma** messaggio per specificare all'interno del codice del programma un messaggio definito dall'utente, utilizzando uno dei formati indicati di seguito.

Se vi è un numero variabile di costanti di stringa, utilizzare:

```
#pragma message( "hi there" )
#pragma message( "hi" " there" )
```

Per scrivere del testo dopo un messaggio, utilizzare:

```
#pragma message text
```

Per espandere un valore definito in precedenza, utilizzare:

```
#pragma message (text)
```

```
#define text "a test string"
#pragma message (text)
```

Ad esempio, per visualizzare questi messaggi su un pulsante, utilizzare:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    #pragma message( "hi there 1" )
    #pragma message( "hi " "there 2" )

    #pragma message hi there 3

    #define text "a test string"
    #pragma message (text)
}
```

Per visualizzare i messaggi nell'IDE, sceglire il comando Projects|Options|Compiler, fare clic sulla pagina Compiler e selezionare la casella di controllo Show general messages.

Inclusione di codice assembler in linea

Se nelle applicazioni Windows si include del codice assembler in linea, potrebbe non essere possibile utilizzare lo stesso codice in Linux a causa dei requisiti PIC position-independent code (PIC) su Linux. Le librerie di oggetti condivisi di Linux (equivalenti alle DLL) richiedono che tutto il codice sia rilocabile in memoria senza modifiche. Ciò influisce principalmente sulle routine assembler in linea che utilizzano variabili globali o altri indirizzi assoluti oppure che chiamano funzioni esterne.

Per le unit che contengono solo codice C++, se necessario, il compilatore genera automaticamente codice PIC. È buona norma compilare ogni file sorgente in entrambi i formati PIC e non-PIC; utilizzare il parametro -VP del compilatore per generare codice PIC.

Si potrebbe voler scrivere le routine assembler in modo diverso a seconda che si stia compilando un eseguibile o una libreria condivisa; utilizzare la direttiva

#ifdef__PIC__ per dividere in due sezioni distinte le due diverse versioni del codice assembler. Oppure, per evitare il problema, si potrebbe riscrivere la routine in C++.

Di seguito sono riportate le regole PIC per il codice assembler in linea:

- PIC richiede che tutti i riferimenti di memoria siano fatti rispetto al registro EBX, che contiene il puntatore all'indirizzo di base del modulo corrente (in Linux viene chiamato Global Offset Table o GOT). Pertanto, invece di

  ```
  MOV EAX,GlobalVar
  ```

  utilizzare

  ```
  MOV EAX,[EBX].GlobalVar
  ```

- PIC richiede che il registro EBX venga protetto durante le varie chiamate delle istruzioni in linguaggio macchina (come in Win32), e inoltre che il registro EBX venga ripristinato prima di effettuare chiamate a funzioni esterne (diversamente da Win32).
- Benché il codice PIC funzioni negli eseguibili di base, potrebbe però diminuire le prestazioni e generare molto codice. Negli oggetti condivisi non vi sono alternative, ma negli eseguibili si vorranno probabilmente ottenere le migliori prestazioni possibili.

Differenze di programmazione in Linux

Il widechar wchar_t di Linux è pari a 32 bit per carattere. Lo standard a 16 bit Unicode supportato dalla VCL e dalla CLX è un sottoinsieme dello standard UCS a 32 bit supportato da Linux e dalle librerie GNU. I tipi WideString devono essere espansi a 32 bit per carattere prima di poterli passare a una funzione del sistema operativo come wchar_t.

In Linux, i tipi WideStrings vengono conteggiati per riferimento analogamente ai tipi long string (a differenza di Windows).

In Windows, i caratteri multibyte (MBCS) sono rappresentati come codici di carattere a uno e a due byte. In Linux, vengono rappresentati da uno a sei byte.

AnsiStrings è in grado di gestire sequenze di caratteri multibyte, dipendenti dalle impostazioni dell'ambiente dell'utente. La codifica Linux per caratteri multibyte come il giapponese, il cinese, l'ebraico e l'arabo potrebbe non essere compatibile con la codifica Windows per lo stesso ambiente. Unicode è portatile, mentre multibyte non lo è. Per i dettagli sulla gestione delle stringhe per le diverse impostazioni locali nelle applicazioni internazionali, consultare "Abilitazione del codice dell'applicazione" a pagina 16-2.

Applicazioni database multipiattaforma

In Windows, C++Builder offre diverse possibilità per accedere alle informazioni di database. Queste includono l'utilizzo di ADO, di BDE (Borland Database Engine) e di InterBase Express. In Windows e Linux è possibile utilizzare *dbExpress*, una

tecnologia multipiattaforma per l'accesso ai dati, a seconda della versione di C++Builder che si posside.

Prima di fare il porting un'applicazione database in *dbExpress* in modo che possa essere eseguita in Linux, si dovrebbero capire le differenze fra l'utilizzo di *dbExpress* e il meccanismo di accesso di dati utilizzato in precedenza. Queste differenze si verificano a vari livelli.

- Al livello più basso, c'è uno strato che comunica tra l'applicazione e il server di database. Questo potrebbe essere ADO, BDE o il software InterBase client. Questo strato è sostituito da *dbExpress*, che è un gruppo di driver leggeri per l'elaborazione dinamica di SQL.
- L'accesso ai dati di basso livello è incluso in un set di componenti che vengono aggiungono ai moduli dati o alle schede. Questi componenti includono i componenti per la connessione ai database, che rappresentano la connessione a un server di database e i dataset, che rappresentano i dati prelevati dal server. Benché vi siano alcune differenze molto importanti, dovute alla natura unidirezionale di cursor di *dbExpress*, le differenze sono meno pronunciate a questo livello, poiché i dataset condividono tutti un antenato comune, come del resto anche i componenti per la connessione ai database.
- A livello interfaccia utente, vi sono ancor meno differenze. I controlli data-aware CLX sono progettati per essere il più simile possibile ai corrispondenti controlli Windows. Le differenze principali a livello di interfaccia utente derivano dalle modifiche necessarie per gestire l'utilizzo degli aggiornamenti in cache.

Per informazioni sul porting di applicazioni database esistenti in *dbExpress*, consultare "Porting di applicazioni database in Linux" a pagina 14-25. Per informazioni sulla progettazione di nuove applicazioni *dbExpress*, vedere il Capitolo 18, "Progettazione di applicazioni database".

Differenze in dbExpress

In Linux, *dbExpress* gestisce la comunicazione con i server di database. *dbExpress* è formato da un gruppo di driver leggeri che implementano un insieme di interfacce comuni. Ogni driver è un oggetto condiviso (file .so) di cui si deve eseguire il link nell'applicazione. Poiché *dbExpress* è progettato per essere multipiattaforma, e disponibile anche in ambiente Windows come un insieme di librerie a collegamento dinamico (.dll).

Come qualsiasi altro strato per l'accesso ai dati, anche *dbExpress* richiede un software sul lato client fornito dal produttore di database. Inoltre, utilizza un driver specifico per il database, più due file di configurazione, dbxconnections e dbxdrivers. Questo è notevolmente meno di quanto sia necessario, ad esempio, per BDE, che richiede la libreria principale di Borland Database Engine (Idapi32.dll) oltre a un driver specifico per il database e a molte altre librerie di supporto.

Inoltre, *dbExpress*:

- consente un percorso più semplice e più rapido per accedere ai database remoti. Di conseguenza è lecito aspettarsi un incremento di prestazioni notevole nel caso di un accesso semplice e diretto ai dati.
- è in grado di elaborare le query e le procedure registrate, ma non supporta il concetto dell'apertura di tabelle.
- restituisce solo cursori unidirezionali.
- non dispone di un supporto interno per l'aggiornamento, se si esclude la capacità di eseguire una query INSERT, DELETE o UPDATE.
- non è in grado di collocare in cache i metadati, e l'interfaccia per l'accesso ai metadati in fase di progettazione è implementata utilizzando l'interfaccia di base per l'accesso ai dati.
- esegue solo le query richieste dall'utente, ottimizzando pertanto l'accesso al database dal momento che non vengono introdotte ulteriori query.
- gestisce internamente un buffer per i record o un blocco di buffer per i record. Questo comportamento è diverso rispetto a BDE, in cui sono i client a dover allocare la memoria utilizzata per memorizzare record.
- supporta solo tabelle locali basate su SQL, come InterBase e Oracle.
- utilizza i driver per DB2, Informix, InterBase, MySQL e Oracle. Se si utilizza un server di database diverso, è necessario convertire i dati in uno di questi formati di database, oppure scrivere un driver *dbExpress* appositamente per il server di database utilizzato oppure, ancora, ottenere un driver *dbExpress* di terze parti per il server di database.

Differenze a livello di componenti

Quando si scrive un'applicazione con *dbExpress*, è necessario utilizzare un set di componenti per l'accesso ai dati diverso da quello utilizzato nelle applicazioni database esistenti. I componenti *dbExpress* condividono le stesse classi di base di altri componenti per l'accesso ai dati (*TDataSet* e *TCustomConnection*), il che significa che molte proprietà, metodi ed eventi sono identici a quelli dei componenti utilizzati nelle applicazioni esistenti.

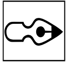
La Tabella 14.9 elenca i principali componenti database utilizzati con InterBase Express, BDE e ADO in ambiente Windows e mostra gli analoghi componenti *dbExpress* da utilizzare in Linux e in applicazioni multipiattaforma.

Tabella 14.9 Componenti paragonabili per l'accesso ai dati

Componenti InterBase Express	Componenti BDE	Componenti ADO	Componenti dbExpress
TIBDatabase	*TDatabase*	*TADOConnection*	*TSQLConnection*
TIBTable	*TTable*	*TADOTable*	*TSQLTable*
TIBQuery	*TQuery*	*TADOQuery*	*TSQLQuery*
TIBStoredProc	*TStoredProc*	*TADOStoredProc*	*TSQLStoredProc*
TIBDataSet		*TADODataSet*	*TSQLDataSet*

I componenti dataset *dbExpress* (*TSQLTable*, *TSQLQuery*, *TSQLStoredProc* e *TSQLDataSet*) sono più limitati dei componenti corrispondenti, in quanto non supportano l'editing e consentono solo lo spostamento in avanti. Per ulteriori informazioni sulle differenze tra i dataset *dbExpress* e gli altri dataset disponibili in Windows, consultare il Capitolo 26, "Uso di dataset unidirezionali".

A causa della mancanza di supporto per l'editing e la navigazione, la maggior parte delle applicazioni *dbExpress* non operano direttamente con i dataset *dbExpress*. Tali applicazioni connettono, invece, il dataset *dbExpress* a un dataset client, che memorizza i record in memoria e fornisce il supporto per l'editing e la navigazione. Per ulteriori informazioni su questa architettura, consultare "Architettura di un database" a pagina 18-6.



Per applicazioni molto semplici, è possibile utilizzare *TSQLClientDataSet* invece di un dataset *dbExpress* connesso a un dataset client. Questo presenta il vantaggio della semplicità, in quanto esiste una corrispondenza 1:1 tra il dataset nell'applicazione di cui si sta facendo il porting e il dataset nell'applicazione convertita, ma risulta meno flessibile rispetto alla connessione esplicita di un dataset *dbExpress* a un dataset client. Per la maggior parte delle applicazioni, si consiglia di utilizzare un dataset *dbExpress* connesso a un componente *TClientDataSet*.

Differenze a livello di interfaccia utente

I controlli data-aware CLX sono progettati per essere il più simile possibile ai corrispondenti controlli Windows. Di conseguenza, il porting dell'interfaccia utente delle applicazioni database richiede poche considerazioni aggiuntive oltre a quelle normalmente implicate nel porting in CLX di una qualsiasi applicazione Windows.

Le principali differenze a livello di interfaccia utente derivano dalle differenze nel modo in cui i dataset *dbExpress* o i dataset client forniscono dati.

Se si utilizzano solo dataset *dbExpress*, è necessario adattare l'interfaccia utente per gestire il fatto che i dataset non supportano l'editing e supportano solo lo spostamento in avanti all'interno del dataset. Pertanto, ad esempio, potrebbe essere necessario rimuovere i controlli che consentono all'utente di passare a un record precedente. Poiché i dataset *dbExpress* non hanno un buffer per i dati, è impossibile visualizzare i dati in una griglia data-aware: è possibile visualizzare solo un record alla volta.

Se invece si connette il dataset *dbExpress* a un dataset client, gli elementi dell'interfaccia utente associati alle funzioni di editing e di navigazione dovrebbero ancora funzionare. Sarà solo necessario riconnetterli al dataset client. La considerazione principale in questo caso è la gestione della modalità di scrittura nel database degli aggiornamenti. Per impostazione predefinita, la maggior parte dei dataset in ambiente Windows scrive automaticamente gli aggiornamenti nel server di database non appena vengono confermati (ad esempio, quando l'utente si sposta su un nuovo record). I dataset client, d'altro canto, conservano sempre gli aggiornamenti nella cache in memoria. Per informazioni su come gestire queste differenze, consultare "Aggiornamento dei dati in applicazioni dbExpress" a pagina 14-28.

Porting di applicazioni database in Linux

Il porting di un'applicazione database in *dbExpress* consente di creare un'applicazione della piattaforma che potrà essere eseguita sia in Windows sia in Linux. Il processo di porting implica alcune modifiche all'applicazione perché la tecnologia è differente. Il grado di difficoltà del porting dipenderà dal tipo di applicazione, dalla sua complessità e dalle operazioni che dovrà compiere. Il porting di un'applicazione che utilizza pesantemente le tecnologie specifiche di Windows, come ADO, sarà più difficile di quello di un'applicazione che utilizza la tecnologia database di C++Builder.

Per eseguire il porting di un'applicazione database Windows/VCL in Linux/CLX, seguire queste direttive generali:

1 Assicurarsi che i dati siano memorizzati in un database supportato da *dbExpress*, come DB2, Informix, InterBase, MySQL e Oracle. I dati devono risiedere in uno di questi server SQL. Se i dati non sono già memorizzati in uno di questi database, trovare un programma di utilità per effettuarne la conversione.

Ad esempio, è possibile utilizzare il programma di utilità Data Pump di C++Builder (non disponibile in tutte le versioni) per convertire alcuni database (come dBase, FoxPro e Paradox) in un database supportato da *dbExpress*. (Per informazioni sull'utilizzo del programma di utilità, consultare il file datapump.hlp in Program Files\Common Files\Borland\Shared\BDE.)

2 Creare moduli dati che contengano i dataset e i componenti connection in modo che risultino separati dalle schede e dai componenti dell'interfaccia utente. In questo modo, si isolano in moduli dati appositi da quelle parti dell'applicazione che richiedono un set completamente nuovo di componenti. Le schede che rappresentano l'interfaccia utente possono essere quindi convertite come una qualsiasi altra applicazione. Per i dettagli, consultare "Porting dell'applicazione" a pagina 14-3.

Le fasi rimanenti partono dal presupposto che i dataset e i componenti connection siano stati isolati nei rispettivi moduli dati.

3 Creare un nuovo modulo dati per contenere le versioni CLX dei componenti dataset e connection.

4 Per ogni dataset nell'applicazione originale, aggiungere un dataset *dbExpress*, un componente *TDataSetProvider* e un componente *TClientDataSet*. Utilizzare le corrispondenze nella Tabella 14.9 per decidere quale dataset *dbExpress* utilizzare. Assegnare a questi componenti nomi significativi.

- Impostare la proprietà *ProviderName* del componente TClientDataSetal nome del componente *TDataSetProvider*.
- Impostare la proprietà *DataSet* del componente TDataSetProvideral dataset *dbExpress*.
- Modificare la proprietà *DataSet* di qualsiasi componente datasource che faceva riferimento al dataset originale in modo che faccia riferimento al dataset client.

5 Impostare le proprietà del nuovo dataset in modo che corrispondano al dataset originale:

- Se il dataset originale era un componente *TTable*, TADOTableo *TIBTable*, impostare la nuova proprietà TableNamedel componente *TSQLTable* al valore di *TableName* del dataset originale. Copiare inoltre tutte le proprietà utilizzate per impostare relazioni master/detail o per specificare indici. Le proprietà utilizzate per specificare intervalli e filtri dovrebbero essere impostate sul dataset client invece che sul nuovo componente *TSQLTable*.
- Se il dataset originale era un componente *TQuery*, TADOQueryo *TIBQuery*, impostare la proprietà SQL del nuovo componente *TSQLQuery* alla proprietà SQL del dataset originale. Impostare la proprietà *Params* del nuovo *TSQLQuery* in modo che corrisponda al valore della proprietà *Params* o *Parameters* del dataset originale. Se era stata impostata la proprietà *DataSource* per stabilire una relazione un master/detail, copiare anche questa.
- Se il dataset originale era un componente *TStoredProc*, TADOStoredProco *TIBStoredProc*, impostare la proprietà *StoredProcName* del nuovo componente *TSQLStoredProc* alla proprietà StoredProcNameo *ProcedureName* del dataset originale. Impostare la proprietà *Params* del nuovo *TSQLStoredProc* in modo che corrisponda al valore della proprietà *Params* o *Parameters* del dataset originale.

6 Per ogni componente connection nell'applicazione originale (*TDatabase*, *TIBDatabase* o *TADOConnection*), aggiungere al nuovo modulo dati un componente *TSQLConnection*. È necessario anche aggiungere un componente *TSQLConnection* per ogni server di database a cui ci si collegava senza utilizzare uno specifico componente connection (ad esempio, utilizzando la proprietà *ConnectionString* di un dataset ADO o impostando la proprietà *DatabaseName* di un dataset BDE a un alias BDE).

7 Per ogni dataset *dbExpress* collocato al punto 4, impostare la rispettiva proprietà *SQLConnection* al componente *TSQLConnection* che corrisponde alla connessione database appropriata.

8 In ogni componente *TSQLConnection*, specificare le informazioni necessarie per stabilire una connessione al database. Per fare ciò, fare doppio clic sul componente *TSQLConnection* per visualizzare il Connection Editor e impostare valori del parametro in modo da indicare le impostazioni appropriate. Se al punto 1 è stato necessario trasferire i dati in un nuovo server di database, allora è necessario specificare le impostazioni appropriate al nuovo server. Se si sta utilizzando lo stesso server usato in precedenza, è possibile cercare alcune di queste informazioni sul componente connection originale:

- Se l'applicazione originale utilizzava *TDatabase*, è necessario trasferire le informazioni contenute nelle proprietà *Params* e *TransIsolation*.
- Se l'applicazione originale utilizzava *TADOConnection*, è necessario trasferire le informazioni contenute nelle proprietà *ConnectionString* e *IsolationLevel*.
- Se l'applicazione originale utilizzava *TIBDatabase*, è necessario trasferire le informazioni contenute nelle proprietà *DatabaseName* e *Params*.

- Se in origine non c’era alcun componente connection, è necessario trasferire le informazioni associate all’alias BDE o ciò che era contenuto nella proprietà *ConnectionString* del dataset.

Si potrebbe decidere di salvare questo set di parametri con un nuovo nome di connessione. Per maggiori informazioni su questa operazione, consultare “Controllo delle connessioni” a pagina 21-3.

Aggiornamento dei dati in applicazioni dbExpress

Le applicazioni *dbExpress* utilizzano i dataset client per supportare l'editing. Quando si confermano le modifiche in un dataset client, le modifiche vengono scritte nella copia dei dati residente in memoria del dataset client ma non vengono riportate automaticamente nel server di database. Se l'applicazione originale utilizzava un dataset client per memorizzare gli aggiornamenti nella cache, non è necessario modificare alcunché per supportare l'editing in Linux. Tuttavia, se si faceva affidamento sul comportamento predefinito della maggior parte dei dataset in Windows, che è quello di scrivere le modifiche nel server di database nel momento in cui vengono confermate, sarà necessario apportare delle modifiche per gestire l'utilizzo di un dataset client.

Esistono due modi per convertire un'applicazione che in precedenza non memorizzava gli aggiornamenti nella cache:

- possibile imitare il comportamento del dataset in Windows scrivendo del codice per applicare ogni record aggiornato al server di database non appena viene confermato. Per fare ciò, fornire al dataset client un gestore di evento *AfterPost* in grado di applicare gli aggiornamenti al server di database:

```
void __fastcall TForm1::ClientDataSet1AfterPost(TDataSet *DataSet)
{
  TClientDataSet *pCDS = dynamic_cast<TClientDataSet *>(DataSet);
  if (pCDS)
    pCDS->ApplyUpdates(1);
}
```

- È possibile adattare l'interfaccia utente in modo che si occupi degli aggiornamenti in cache. Questo approccio presenta alcuni vantaggi, come la riduzione delle quantità di traffico di rete e la riduzione dei tempi di transazione. Tuttavia, se si passa all'utilizzo di aggiornamenti in cache, occorre decidere quando applicare quegli aggiornamenti al server di database e probabilmente apportare delle modifiche all'interfaccia utente per consentire agli utenti di iniziare l'applicazione degli aggiornamenti o per informarli se le modifiche apportate sono stato registrate nel database. Inoltre, poiché quando l'utente invia un record gli errori di aggiornamento non vengono rilevati, sarà necessario modificare il modo in cui vengono segnalati tali errori all'utente, affinché possano vedere non solo quale aggiornamento ha causato un problema ma anche che tipo di problema si è verificato.

Se l'applicazione originale utilizzava il supporto fornito da BDE o da ADO per aggiornamenti in cache, sarà necessario apportare alcune modifiche nel codice per passare all'utilizzo di un dataset client. La tabella seguente elenca proprietà, eventi e

metodi che supportano gli aggiornamenti in cache con i dataset BDE e ADO e proprietà, metodi ed eventi corrispondenti nei *TClientDataSet*:

Tabella 14.10 Proprietà , metodi ed eventi degli aggiornamenti in cache

Nei dataset BDE (o TDatabase)	Nei dataset ADO	In TClientDataSet	Funzione
CachedUpdates	*LockType*	Non necessario, i dataset client memorizzano sempre gli aggiornamenti in cache.	Determina se sono attivi gli aggiornamenti in cache.
Non supportato	*CursorType*	Non supportato	Specifica in che modo è isolato il dataset rispetto alle modifiche sul server.
UpdatesPending	Non supportato	*ChangeCount*	Indica se la cache locale contiene record aggiornati che devono essere applicati al database.
UpdateRecordTypes	*FilterGroup*	*StatusFilter*	Indica il tipo di record aggiornati da rendere visibile quando si applicano gli aggiornamenti in cache.
UpdateStatus	*RecordStatus*	*UpdateStatus*	Indica se un record è invariato, modificato, inserito o cancellato.
OnUpdateError	Non supportato	*OnReconcileError*	Un evento per la gestione degli errori di aggiornamento record-per-record.
ApplyUpdates (in dataset o database)	*UpdateBatch*	*ApplyUpdates*	Applica al database i record nella cache locale.
CancelUpdates	*CancelUpdates* o *CancelBatch*	*CancelUpdates*	Rimuove dalla cache locale gli aggiornamenti in sospeso senza applicarli.
CommitUpdates	Gestito automaticamente	*Reconcile*	Cancella la cache di aggiornamento dopo che l'applicazione degli aggiornamenti è andata a buon fine.
FetchAll	Non supportato	*GetNextPacket* (e *PacketRecords*)	Copia i record del database nella cache locale per l'editing e l'aggiornamento.
RevertRecord	*CancelBatch*	*RevertRecord*	Annulla gli aggiornamenti al record corrente nel caso gli aggiornamenti non siano ancora stati applicati.

Applicazioni Internet multipiattaforma

Un'applicazione Internet è un'applicazione client/server che utilizza i protocolli standard Internet per connettere il client al server. Poiché l'applicazione utilizza i protocolli standard Internet per le comunicazioni client/server, è possibile rendere le applicazioni indipendenti dalla piattaforma. Ad esempio, un programma lato server per un'applicazione Internet comunica con il client tramite il software per Web server della macchina. L'applicazione server è di solito scritta per Linux o per Windows, ma può anche essere multipiattaforma. I client possono risiedere su entrambe le piattaforme.

C++Builder consente di creare applicazioni per Web server sotto forma di applicazioni CGI o Apache da distribuire in ambiente Linux. In Windows, è possibile creare altri tipi di server Web come DLL di Microsoft Server (ISAPI), DLL di Netscape Server (NSAPI) o applicazioni CGI Windows. Solo le applicazioni prettamente CGI e alcune applicazioni che utilizzano WebBroker potranno essere eseguite sia in Windows sia in Linux.

Porting di applicazioni Internet in Linux

Se si hanno applicazioni Internet esistenti che si desidera trasformare in applicazioni multipiattaforma, è possibile o fare il porting dell'applicazione per Web server in Linux o creare una nuova applicazione in Linux quando è disponibile una soluzione di C++ di Borland. Per informazioni sulla scrittura di server Web, consultare il Capitolo 32, "Creazione di applicazioni server per Internet". Se l'applicazione utilizza WebBroker e scrive sull'interfaccia WebBroker e non utilizza chiamate API native, non sarà molto difficile eseguire il porting in Linux.

Se l'applicazione scrive su ISAPI, NSAPI, Windows CGI o altre API per Web, sarà molto più difficile fare il porting. Sarà necessario ricercare queste chiamate API nei file sorgente e tradurle in chiamate Apache (consultare ..\Include\Vcl\httpd.hpp nella directory Internet per i prototipi delle funzione delle API Apache) o in chiamate CGI. È necessario anche apportare tutte le altre modifiche suggerite descritte nel paragrafo "Porting di applicazioni Windows in Linux" a pagina 14-2.

Capitolo

15

Uso di package e di componenti

Un *package* è una particolare libreria a collegamento dinamico che può essere usata dalle applicazioni C++Builder, dall'IDE o da entrambi. I *package di esecuzione* forniscono funzionalità quando l'utente esegue un'applicazione. I *package di progettazione* vengono usati per installare componenti nell'IDE e per creare speciali editor di proprietà per componenti custom. Un singolo package può funzionare sia per la progettazione sia per l'esecuzione e i package di progettazione spesso funzionano chiamando i package di esecuzione. Per distinguerle dalle altre DLL, le librerie di package vengono memorizzate in file con estensione .bpl (Borland package library.

Come le altre librerie runtime, i package contengono codice che può essere condiviso tra le applicazioni. Ad esempio, i componenti C++Builder usati più frequentemente risiedono in un package chiamato vcl. Ogni volta che si crea una nuova applicazione predefinita, essa utilizza automaticamente la vcl. Quando si compila un'applicazione creata in questo modo, l'immagine dell'eseguibile dell'applicazione contiene solo il codice e i dati che lo caratterizzano; il codice comune si trova nel package di esecuzione di nome vcl60.bpl. Un computer sui cui sono state installate diverse applicazioni abilitate all'uso del package ha bisogno di una sola copia del file vcl60.bpl, che viene condiviso da tutte le applicazioni e anche dall'IDE stesso di C++Builder.

C++Builder viene distribuito con numerosi package di esecuzione precompilati che incapsulano i componenti della VCL e della CLX. C++Builder usa i package di progettazione anche per trattare i componenti nell'IDE.

È possibile costruire applicazioni con o senza package. Tuttavia, se si vogliono aggiungere componenti personalizzati all'IDE, occorre installarli come package di progettazione.

I package di esecuzione possono essere creati in modo da poterli condividere fra le applicazioni. Se si scrivono componenti per C++Builder, è possibile compilarli creando dei package di progettazione prima di installarli.

Perché usare i package

I package di progettazione semplificano le operazioni di distribuzione e di installazione dei componenti custom. I package di esecuzione, che sono facoltativi, offrono diversi vantaggi rispetto alla programmazione convenzionale. Compilando del codice riutilizzato in una libreria runtime, è possibile condividerlo tra varie applicazioni. Per esempio, tutte le applicazioni—compreso lo stesso C++Builder—possono accedere ai componenti standard tramite package. Poiché le applicazioni non hanno nei loro eseguibili copie distinte della libreria di componenti, gli eseguibili stessi risultano molto più piccoli, risparmiando risorse di sistema e spazio sul disco rigido. Inoltre, i package consentono una compilazione più rapida, in quanto ad ogni compilazione viene compilato solo il codice che caratterizza la specifica applicazione.

Package e DLL standard

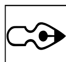
Quando si vuole ottenere un componente custom che sia disponibile tramite l'IDE di C++Builder, si deve creare un package. Quando, invece, si desidera costruire una libreria che possa essere chiamata da qualsiasi applicazione, indipendentemente dallo strumento di sviluppo usato per costruire l'applicazione, si deve creare una DLL standard.

La tabella seguente elenca i tipi di file associati con i package:

Tabella 15.1 File package

Estensione del file	Contenuto
bpk	Il file sorgenet delle opzioni di progetto. Questo file è la parte XML del progetto del package. *ProjectName*.bpk e *ProjectName*.cpp vengono usati in combinazione per gestire le impostazioni, le opzioni e i file utilizzati dal progetto del package.
bpl	Il package di esecuzione. Questo file è una DLL di Windows con caratteristiche specifiche per C++Builder. Il nome base del file .bpl è il nome base del file sorgente bpk.
cpp	*ProjectName*.cpp contiene il punto d'ingresso del package. Solitamente, inoltre, ogni componente incluso nel package risiede in un file.cpp.
h	Il file header o interfaccia per il componente. *ComponentName*.h è il file complementare di *ComponentName*.cpp.
lib	Una libreria statica, o collection di .obj, usata al posto di un .bpi nel caso in cui l'applicazione non usi package di esecuzione. Viene generata solo se è selezionata l'opzione -Gl (Generate .LIB File).
obj	Un'immagine binaria di un file unit contenuto in un package. Viene creato, se occorre, un file obj per ciascun file .cpp.
bpi	Una libreria di importazione di package Borland (Borland package import library). Per ogni package viene creato un singolo file .bpi. I file bpi nel caso delle bpl sono analoghi alle librerie di importazione per le dll. Questo file viene passato al linker file dalle applicazioni che usano il package per risolvere i riferimenti a funzioni presenti nel package. Il nome base del bpi è uguale al nome base del file sorgente del package.

In un package è possibile includere componenti VCL o componenti CLX o entrambi. I package destinati all'uso su più piattaforme dovrebbero includere solo componenti CLX.



In un'applicazione i package condividono i loro dati globali con gli altri moduli.

Package di esecuzione

I package di esecuzione vengono distribuiti con le applicazioni C++Builder. Essi forniscono funzionalità quando un utente esegue l'applicazione.

Per eseguire un'applicazione che usa i package, un computer deve contenere sia il file .EXE dell'applicazione sia tutti i package (i file .bpl) che essa utilizza. Affinché un'applicazione sia in grado di usare i file .bpl, tali file devono essere nel percorso di ricerca del sistema. Quando si distribuisce un'applicazione, occorre accertarsi che gli utenti abbiano le versioni corrette di tutte i file .bpl necessari.

Uso dei package in un'applicazione

Per usare i package in un'applicazione:

1. Caricare o creare un progetto nell'IDE.
2. Scegliere Project|Options.
3. Scegliere il separatore Packages.
4. Selezionare la casella di controllo "Build with Runtime Packages" e immettere uno o più nomi di package nella casella di testo sottostante. (I package di esecuzione associati ai package di progettazione installati sono già elencati nella casella di testo.)
5. Per aggiungere un package ad un elenco esistente, fare clic sul pulsante Add e immettere il nome del nuovo package nella finestra di dialogo Add Runtime Package. Per consultare l'elenco di package disponibili, fare clic sul pulsante Add, quindi, nella finestra di dialogo Add Runtime Package, fare clic sul pulsante Browse accanto alla casella di testo Package Name.

   Se si modifica la casella di testo Search Path della finestra di dialogo Add Runtime Package, viene cambiato il Library Path globale di C++Builder.

   Non è necessario includere l'estensione del file nei nomi dei package (né tantomeno il numero che rappresenta la versione di C++Builder); cioè, vcl60.bpl viene scritto come vcl. Se si scrive direttamente nella casella di testo Runtime Packages, bisogna ricordarsi di separare i nomi con il punto e virgola. Per esempio:

   ```
   rtl;vcl;vcldb;vclado;vclx;vclbde;
   ```

I package elencati nella casella di testo Runtime Packages vengono collegati automaticamente all'applicazione. I nomi di package duplicati vengono ignorati, e se

la casella di controllo Build with runtime packages è deselezionata, viene eseguito il link dell'applicazione senza package.

I package di esecuzione vengono selezionati solo per il progetto attivo. Per trasformare le scelte effettuate in impostazioni predefinite automatiche per i nuovi progetti, selezionare la casella di controllo Defaults nella zona inferiore della finestra di dialogo.

Un'applicazione creata con package deve includere anch'essa i file header per le unit incluse nel package che essa utilizza. Per esempio, un'applicazione che utilizza i controlli per database richiede l'istruzione

```
#include "vcldb.h"
```

anche nel caso utilizzi il package vcldb. Nei file sorgenti generati, C++Builder crea automaticamente queste istruzioni **#include**.

Caricamento dinamico dei package

Per caricare un package in esecuzione, chiamare la funzione *LoadPackage*. *LoadPackage* carica il package, controlla eventuali unit duplicate e chiama i blocchi di inizializzazione di tutte le unit contenute nel package. Ad esempio, quando si sceglie un file in una finestra di dialogo per la selezione di file, potrebbe essere eseguito il codice seguente.

```
if (OpenDialog1->Execute())
  PackageList->Items->AddObject(OpenDialog1->FileName, (TObject *)LoadPackage(OpenDialog1-
>FileName));
```

Per scaricare dinamicamente un package, chiamare *UnloadPackage*. Occorre ricordarsi di distruggere qualsiasi istanza delle classi definite nel package e di deregistrare tutte quelle classi che sono state registrate dal package.

Scelta di quali package di esecuzione usare

C++Builder viene fornito con diversi package di esecuzione, compresi rtl e vcl, che forniscono il supporto per il linguaggio di base e per i componenti.

Il package vcl contiene i componenti usati più di frequente; il package rtl include tutte le funzioni di sistema non incluse in componenti e gli elementi dell'interfaccia Windows. Non include i componenti database, o altri speciali componenti, che sono disponibili in altri package.

Per creare un'applicazione database client/server che utilizza i package, occorrono diversi package di esecuzione, compresi vcl, vcldb,rtl e dbrtl. Se nell'applicazione si vogliono usare componenti Outline, occorre anche vclx. Per usare questi package, scegliere Project|Options, selezionare il separatore Packages e immettere il seguente elenco nella casella di testo Runtime Packages.

```
rtl;vcl;vcldb;vclx;
```

In realtà, non è necessario includere vcl ed rtl, in quanto tali package sono referenziati nella lista Requires di vcldb. (Vedere l'argomento "Elenco Requires" a

pagina 15-10.) L'applicazione sarà compilata esattamente allo stesso modo, indipendentemente dal fatto che i package vcl e rtl siano inclusi o meno nella casella di testo Runtime Packages.

Package custom

Un package custom è un file bpl scritto e compilato dall'utente oppure un package precompilato fornito da un produttore terzo. Per usare in un'applicazione un package custom di esecuzione, scegliere Project|Options ed aggiungere alla casella di testo Runtime Packages nella pagina Packages il nome del package. Per esempio, si supponga di avere un package di statistica denominato stats.bpl. Per usarlo in un'applicazione, la riga da immettere nella casella di testo Runtime Packages sarebbe la seguente:

```
rtl;vcl;vcldb;stats
```

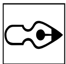
Se sono stati creati dei package personalizzati, se necessario è possibile aggiungerli all'elenco.

Package di progettazione

I package di progettazione vengono usati per installare i componenti nella Component palette dell'IDE e per creare speciali editor di proprietà per i componenti custom.

C++Builder viene fornito con i seguenti package di progettazione preinstallati nell'IDE. I package installati dipendono dalla versione di C++Builder utilizzata e dal fatto che l'IDE sia stato personalizzato o meno. È possibile vedere una lista dei package installati nel sistema scegliendo il comando Component|Install Packages.

I package di progettazione funzionano chiamando i package di esecuzione che vengono referenziati nei rispettivi elenchi Requires. (Vedere l'argomento "Elenco Requires" a pagina 15-10.) Per esempio, dclstd contiene riferimenti alla vcl. Lo stesso Dclstd contiene un'ulteriore funzionalità che rende disponibile nella Component palette la maggior parte dei componenti standard.



Per convenzione, i package di progettazione dell'IDE design iniziano con dcl e risiedono nella directory bin. Nel caso di package di progettazione, quale ad esempio ..\bin\Dclstd, esistono le rispettive controparti per il linker, compresi ..\lib\vcl.lib, ..\lib\vcl.bpi, e lo stesso package di esecuzione, Windows\System\vcl60.bpl.

Oltre ai package preinstallati, è possibile installare nell'IDE propri package di componenti, o package di componenti di sviluppatori terzi. Il package di progettazione dclusr viene fornito come contenitore predefinito per nuovi componenti.

Installazione di package di componenti

Tutti i componenti vengono installati nell'IDE come package. Nel caso siano stati scritti componenti personali, occorre creare e compilare un package che li contenga. Vedere "Creazione ed editing dei package" a pagina 15-7.) Il codice sorgente del componente deve seguire il modello descritto nella Parte V, "Creazione di componenti custom". Se si stanno aggiungendo più unit a un singolo package, e ognuna di esse contiene componenti, è necessario creare una singola funzione Register per tutti i componenti in un namespace che abbia il nome del package.

Per installare o disinstallare i propri componenti personali o quelli di produttori terzi:

1 Se si sta installando un nuovo package, copiare o spostare i file package in una directory locale. Se il package è stato fornito con i file .bpl, .bpi, .lib e .obj, accertarsi di copiarli tutti. (Per informazioni su questi file, vedere "Package e DLL standard".)

   La directory in cui vengono memorizzati il file .bpi e i file header, e i file .lib o .obj se inclusi nella distribuzione, deve risultare nel Library Path di C++Builder.

2 Scegliere Component | Install Packages dal menu dell'IDE, oppure scegliere Project | Options e fare clic sul separatore Packages.

3 Sotto "Design packages" appare un elenco dei package disponibili.

   - Per installare un package nell'IDE, selezionare la casella di controllo posta a fianco.
   - Per disinstallare un package, deselezionare la relativa casella di controllo.
   - Per vedere l'elenco dei componenti inclusi in un package installato, selezionare il package e fare clic su Components.
   - Per aggiungere un package all'elenco, fare clic su Add e ricercare nella finestra Add Design Package la directory in cui risiede il file .BPL (vedere il punto 1). Selezionare un file .bpl e fare clic su Open.
   - Per rimuovere un package dall'elenco, selezionarlo e fare clic su Remove.

4 Fare clic su OK.

I componenti nel package vengono installati nelle pagine della Component palette specificate nella procedura *RegisterComponents* dei componenti, usando i nomi a loro assegnati nella stessa procedura.

I nuovi progetti vengono creati utilizzando tutti i package disponibili installati, a meno che non si modifichino le impostazioni predefinite. Per fare sì che le opzioni di installazione correnti diventino quelle predefinite per tutti i nuovi progetti, selezionare la casella di controllo Default nella parte inferiore della pagina Packages della finestra di dialogo Project Options.

Per rimuovere i componenti dalla Component palette senza disinstallare un package, selezionare Component | Configure Palette, oppure Tools | Environment Options e fare clic sul separatore Palette. Il separatore Palette options elenca ciascun componente installato insieme al nome della pagina della Component palette in cui

esso appare. Selezionando un componente e facendo clic su Hide, il componente viene rimosso dalla tavolozza.

Creazione ed editing dei package

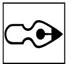
La creazione di un package richiede che si specifichi:

- Un *nome* per il package.
- Un elenco degli altri package che sono *richiesti* dal nuovo package, o che sono ad esso collegati.
- Un elenco dei file unit che sono *inglobati* o collegati al package al momento della compilazione. Il package è essenzialmente un wrapper per queste unit di codice sorgente. L'elenco Contains indica il luogo in cui vengono collocate le unit di codice sorgente per i componenti custom che devono essere compilati in un package.

Un package è definito da un file sorgente C++ (.cpp) e da un file di opzioni di progetto il cui nome termina con l'estensione .bpk. Questi file sono generati dal Package editor.

Creazione di un package

Per creare un package, seguire la procedura riportata di seguito. Per maggiori informazioni sui punti qui descritti, consultare "Struttura di un package" a pagina 15-9.



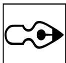
Non utilizzare direttive ifdef in un file package (.bpk) come si fa quando si sviluppano applicazioni multipiattaforma. È possibile, tuttavia, utilizzarle nel codice sorgente.

1. Scegliere File | New | Other, selezionare l'icona Package e fare clic su OK.
2. Il package generato viene visualizzato nel Package editor.
3. Il Package editor mostra un nodo *Requires* e un nodo *Contains* per il nuovo package.
4. Per aggiungere una unit all'elenco Contains, fare clic sul pulsante Add to package. Nella pagina Add unit, scrivere nella casella di testo Unit file name il nome di un file .cpp, oppure fare clic su Browse per ricercare il file, quindi fare clic su OK. La unit selezionata apparirà sotto il nodo Contains nel Package editor. È possibile aggiungere altre unit ripetendo la procedura.
5. Per aggiungere una unit all'elenco Requires, fare clic sul pulsante Add to package. Nella pagina Requires, scrivere nella casella di testo Package name il nome di un file .bpi, oppure fare clic su Browse per ricercare il file, quindi fare clic su OK. Il package selezionato appare sotto il nodo Requires nell'editor di Package. È possibile aggiungere altri package ripetendo questo operazione.

6 Fare clic sul pulsante di scelta rapida Options e decidere il tipo di package che si vuol generare.

- Per creare un package utilizzabile solo in fase di progettazione (un package che non può essere usato in fase di esecuzione), selezionare il pulsante di opzione Designtime only. (Oppure aggiungere l'opzione **-Gpd** del linker al file .bpk: `LFLAGS = ... -Gpd ....`)
- Per creare un package utilizzabile solo in fase di esecuzione (un package che non può essere installato), selezionare il pulsante di opzione Runtime only. (Oppure aggiungere l'opzione **-Gpr** del linker al file .bpk: `LFLAGS = ... -Gpr ....`)
- Per creare un package disponibile in fase di progettazione e di esecuzione, selezionare il pulsante di opzione Designtime and runtime.

7 Per compilare il package, fare clic nel Package editor sul pulsante di scelta rapida Compile package.



È possibile anche fare clic sulpulsante Install per forzare l'esecuzione del make.

Editing di un package esistente

È possibile aprire un package esistente per apportavi delle modifiche in diversi modi:

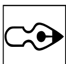
- Scegliere File|Open (o File|Reopen) e selezionare un file cpp o bpk.
- Scegliere Component|Install Packages, selezionare un package dall'elenco Design Packages e fare clic sul pulsante Edit.
- Non appena l'editor di Package è aperto, selezionare uno dei package nel nodo Requires, fare clic-destro e scegliere Open.

Per modificare la descrizione di un package o impostare le opzioni relative all'uso, fare clic nell'editor di Package sul pulsante di scelta rapida Options e selezionare il separatore Description.

La finestra di dialogo Project Options ha una casella di controllo Default nell'angolo in basso a sinistra. Se si fa clic su OK quando questa casella è selezionata, le opzioni scelte vengono salvate come impostazioni predefinite per nuovi package di progetto. Per ripristinare le impostazioni predefinite originali, cancellare o cambiare nome al file default.bpk.

File sorgenti di package e file di opzione di progetto

I file sorgenti di package hanno estensione .cpp. I file di opzioni di progetto dei package vengono creati utilizzando il formato XML ed hanno l'estensione .bpk (Borland package). Il file di opzione di progetto del package viene visualizzato dall'editor dei package facendo clic destro sulla clausola *Contains* o *Requires* e scegliendo il comando Edit Source Option.



C++Builder esegue la manutenzione del file .bpk. Solitamente non è necessario modificarlo manualmente. Le modifiche dovrebbero essere apportate utilizzando la pagina Packages della finestra di dialogo Project Options.

Il file di opzione di progetto per un package di nome MyPack dovrebbe assomigliare, almeno in parte, al seguente:

```
<MACROS>
    <VERSION value="BCB.05.02"/>
    <PROJECT value="MyPack.bpl"/>
    <OBJFILES value="MyPack.obj Unit2.obj Unit3.obj"/>
    <RESFILES value="MyPack.res"/>
    <IDLFILES value=""/>
    <IDLGENFILES value=""/>
    <DEFFILE value=""/>
    <RESDEPEN value="$(RESFILES)"/>
    <LIBFILES value=""/>
    <LIBRARIES value=""/>
    <SPARELIBS value="Vcl60.lib"/>
    <PACKAGES value="Vcl60.bpi vcldbx60.bpi"/>
     .
     .
     .
```

In questo caso, MYPACK.cpp dovrebbe includere il codice seguente:

```
USERES("MyPack.res");
USEPACKAGE("vcl60.bpi");
USEPACKAGE("vcldbx60.bpi");
USEUNIT("Unit2.cpp");
USEUNIT("Unit3.cpp");
```

L’elenco Contains di MyPack include tre unit: la stessa MyPack, Unit2 e Unit3. L’elenco Requires di MyPack include VCL e VCLDBX.

Packaging di componenti

Se si utilizza il New Component wizard per creare componenti (scegliendo il comando Component|New Component), ove necessario, C++Builder inserisce la macro PACKAGE. Ma se si dispone di componenti custom provenienti da una versione precedente di C++Builder, è necessario aggiungere manualmente la macro PACKAGE in due punti.

La dichiarazione del file header per un componente C++Builder deve includere la macro predefinita PACKAGE dopo la parola **class**:

```
class PACKAGE MyComponent : ...
```

E nel file cpp in cui è definito il componente, includere la macro PACKAGE nella dichiarazione della funzione *Register*:

```
void __fastcall PACKAGE Register()
```

La macro PACKAGE viene espansa in un’istruzione che consente di importare ed esportare le classi dal file .bpl risultante.

Struttura di un package

I package sono composti dalle parti seguenti:

- Nome del package
- Elenco Requires
- Elenco Contains

Assegnazione del nome ai package

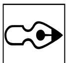
All'interno di un progetto, i nomi del package devono essere univoci. Se a un package viene assegnato il nome Stats, il Package editor genera un file sorgente e un file di opzione di progetto di nome, rispettivamente, Stats.cpp e Stats.bpk; il compilatore genera un eseguibile, un'immagine binaria e (facoltativamente) una libreria statica di nome Stats.bpl, Stats.bpi e Stats.lib. Per utilizzare il package in un'applicazione, aggiungere Stats alla casella di testo Runtime Packages (dopo aver scelto il comando Project|Options e aver fatto clic sulla pagina Package).

Al nome del package è possibile anche aggiungere un prefisso, un suffisso e un numero di versione. Mentre il Package editor è aperto, fare clic sul pulsante Options. Sulla pagina Description della finestra di dialogo Project Options, imettere del testo o un valore per le opzioni LIB Suffix, LIB Prefix, o LIB Version. Ad esempio, per aggiungere un numero di versione al progetto del package, immettere 6 dopo LIB Version in modo che *Package1* generi *Package1*.bpl.6.

Elenco Requires

L'elenco Requires specifica gli altri package esterni usati dal package attivo. Un package esterno incluso nell'elenco Requires viene collegato automaticamente in fase di compilazione a qualsiasi applicazione che usi il package attivo e una delle unit contenute nel package esterno.

Se i file unit contenuti nel package fanno riferimento ad altre unit presenti in altri package, questi package dovrebbero apparire nell'elenco Requires oppure, nel caso non fossero presenti, li si dovrebbe aggiungere. Se gli altri package vengono omessi dall'elenco Requires, il compilatore li importerà nel package come 'unit contenute implicitamente.'



La maggior parte dei package che vengono creati richiedono rtl. Se si utilizzano componenti VCL, sarà necessario includere anche il package vcl. Se si utilizzano componenti CLX per programmazione multipiattaforma, è necessario includere VisualCLX.

Evitare riferimenti circolari a package

I package non possono contenere riferimenti circolari nell'elenco *Requires*. Ciò significa che

- Un package non può fare riferimento a se stesso nel proprio elenco Requires.
- Una catena di riferimenti deve terminare senza fare riferimento ad alcun package della catena. Se il package A richiede il package B, allora il package B non può richiedere il package A; se il package A richiede il package B e il package B richiede il package C, allora il package C non può richiedere il package A.

Gestione dei riferimenti duplicati del package

I riferimenti duplicati nell'elenco Requires del package, o nella casella di testo Runtime Packages, vengono ignorati dal compilatore. Per chiarezza di programmazione e leggibilità, tuttavia, è opportuno individuare e rimuovere tali riferimenti.

Elenco Contains

L'elenco Contains identifica i file unit che devono essere inclusi nel package. Se si sta scrivendo un package personale, immettere il codice sorgente in file cpp e includerli nell'elenco Contains.

Evitare uses ridondanti nel codice sorgente

Un package non può apparire nell'elenco Contains di un altro package.

Tutte le unit incluse direttamente nell'elenco Contains di un package, o incluse indirettamente in una di quelle unit, vengono incorporate nel package in fase di collegamento.

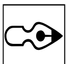
Una unit non può essere contenuta (direttamente o indirettamente) in più di un package usato dalla stessa applicazione, *incluso l'IDE di C++Builder*. Ciò significa che se si crea un package che contiene una delle unit in vcl, non sarà possibile installare il package nell'IDE. Per usare un file unit già inserito in un altro package, mettere il primo package nell'elenco Requires del secondo package.

Generazione di package

È possibile compilare un package dall'IDE o dalla riga comandi. Per ricompilare automaticamente un package dall'IDE:

1 Scegliere File | Open, selezionare un file sorgente di package o un file di opzioni di progetto e fare clic su Open.

2 Non appena si apre l'editor, scegliere il comando Project | Make o Project | Build.



È possibile anche scegliere il comando File | New | Other e fare doppio click su Package editor. Fare clic sul pulsante Install per eseguire il make del progetto del package. Fare clic destro sui nodi del progetto del package per selezionare le opzioni per installare, eseguire il make o il build del package.

Nel caso si stia aggiungeno un file .cpp non generato in automatico, aggiungere al codice sorgente del package una delle direttive del compilatore. Per maggiori informazioni, vedere la sezione "Direttive al compilatore specifiche per il packaging".

Quando si esegue il link dalla riga comandi, è possibile utilizzare diverse opzioni del linker specifiche per i package. Per maggiori informazioni, consultare "Uso del compilatore e del linker dalla riga comandi" a pagina 15-13.

Direttive al compilatore specifiche per il packaging

La tabella seguente elenca le direttive al compilatore specifiche per il packaging che è possibile inserire nel codice sorgente.

Tabella 15.2 Direttive al compilatore specifiche per il packaging

Direttiva	Scopo
#pragma package(smart_init)	Assicura che le unit inserite nel package vengano inizializzate nell'ordine determinato in base alle rispettive dipendenze. (Inclusa per impostazione predefinita nel file sorgente del package.)
#pragma package(smart_init, weak)	Incorpora "debolmente" una unit nel package. Consultare la sezione "Packaging debole".(Direttiva inclusa nel file sorgente della unit.)

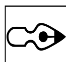
Fare riferimento al paragrafo "Creazione di package e di DLL" a pagina 7-10 per uteriori direttive utilizzabili in tutte le librerie.

Packaging debole

La direttiva **#pragma package (smart_init, weak)** influisce sul modo in cui un file .obj viene incorporato nei file .bpi e .bpl del package. (Per informazioni sui file generati dal compilatore, vedere il paragrafo "File package creati mediante generazione" a pagina 15-13.) Se in un file di unit appare la direttiva **#pragma package(smart_init, weak)**, il compilatore, se possibile, omette la unit dalle BPL, e crea una copia locale della unit non incorporata nel package nel caso questa venga richiesta da un'altra applicazione o da un altro package. Le unit compilate in questo modo vengono definite come *weakly packaged*.

Si supponga, ad esempio, di aver creato un package chiamato PACK contenente una sola unit, UNIT1. Si supponga che UNIT1 non faccia uso di altre unit, ma che faccia delle chiamate a RARE.dll. Se quando si compila il package, si inserisce la direttiva **#pragma package(smart_init, weak)** in UNIT1.cpp, UNIT1 non verrà inclusa in PACK.dpl. Non si dovranno distribuire copie di RARE.dll insieme a PACK. Tuttavia, UNIT1 continuerà a essere inclusa in PACK.bpi. Se UNIT1 viene referenziata da un altro package o da un'altra applicazione che utilizzano PACK, questa verrà copiata da PACK.bpi e compilata direttamente nel progetto.

Ora, si supponga di aggiungere a PACK una seconda unit, UNIT2. Si supponga anche che UNIT2 utilizzi UNIT1. Questa volta, anche se si compila PACK con la direttiva **#pragma package(smart_init, weak)** in UNIT1.cpp, il compilatore includerà UNIT1 in PACK.bpl. Invece, gli altri package e le altre applicazioni che fanno riferimento a UNIT1 utilizzeranno la copia (non inserita nel package) presa da PACK.bpi.



I file di unit contenenti la direttiva **#pragma package(smart_init, weak)** non devono avere variabili globali.

La direttiva **l#pragma package(smart_init, weak)** è una funzionalità avanzata destinata agli sviluppatori che distribuiscono le proprie bpl ad altri programmatori in C++Builder. Evita la distribuzione di DLL utilizzate raramente ed elimina i conflitti tra package che dipendono dalla stessa libreria esterna.

Ad esempio, la unit PenWin di C++Builder fa riferimento a PenWin.dll. Pochi progetti utilizzano PenWin e nella maggior parte degli elaboratori PenWin.dll non è installata. Per questo motivo, la unit PenWin è incorporata "debolmente"nella vcl. Quando si compila un progetto che utilizza PenWin e il package vcl, PenWin viene copiata da vcl60.bpi e incorporata direttamente nel progetto; il file eseguibile risultante è collegato in modo statico a PenWin.dll.

Se PenWin non fosse incorporata debolmente, sorgerebbero due problemi. Primo, la stessa vcl sarebbe collegata in modo statico a PenWin.dll, per cui non la si potrebbe caricare in quei computer in cui PenWin.dll non è installata. Secondo, se si tentasse di creare un package contenente PenWin, si otterrebbe un errore del compilatore perché la unit PenWin sarebbe contenuta sia in vcl che nel package. Di conseguenza, in assenza di un packaging debole, non si potrebbe includere PenWin nelle distribuzioni standard della vcl.

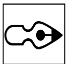
Uso del compilatore e del linker dalla riga comandi

Quando si compila dalla riga comandi, utilizzare l'opzione **-Tpp** del linker per assicurarsi che il progetto venga generato come package. Altre opzioni specifiche per i package sono elencate nella tabella seguente.

Tabella 15.3 Opzioni del linker della riga comandi specifiche per i package

Opzione	Scopo
-Tpp	Genera il progetto come package. Inclusa per impostazione predefinita nei file di progetto dei package.
-Gi	Salva il file bpi generato. Inclusa per impostazione predefinita nei file di progetto dei package.
-Gpr	Genera un package da usare solo in esecuzione.
-God	Genera un package da usare solo in progettazione.
-Gl	Genera un file .lib.
-D "*description*"	Salva la descrizione specificata con il package.

Le opzioni -**Gpr** e -**Gpd** corrispondono alle caselle di controllo Runtime Package e Design Package presenti sulla pagina Description della finestra di dialogo Project Options (disponibile solo per i progetti di package); se non si utilizza né -**Gpr** né -**Gpd**, il package risultante funziona sia in progettazione sia in esecuzione. L'opzione -**D** corrisponde alla casella di testo Description sulla stessa pagina. L'opzione **-Gl** corrisponde alla casella di controllo Generate .lib File presente sulla pagina Linker della finestra di dialogo Project Options.



È possibile generare un file make da utilizzare dalla riga comandi scegliendo il comando Project | Export Makefile.

File package creati mediante generazione

Per creare un package, si compila un file sorgente (.cpp) che utilizza un file di opzioni di progetto con estensione .bpk. Il nome base del file sorgente dovrebbe coincidere col nome base dei file generato dal compilatore; ovvero, se il file sorgente si chiama TRAYPAK.cpp, il file di opzioni di progetto—TRAYPAK.bpk— dovrebbe includere

```
<PROJECT value="Traypak.bpl"/>
```

In questo caso, compilando il progetto verrà creato un package di nome TRAYPAK.bpl.

Quando si esegue la compilazione e il link di un package, si crea un file bpi, un bpl, un obj, e un lib. Quando li si compila, per impostazione predefinita i file bpi, bpl e di libreria vengono generati nelle directory specificate nella pagina Library della finestra di dialogo Tools | Environment Options. È possibile ridefinire le impostazioni predefinite facendo clic sul pulsante di scelta rapida Options nell'editor di package in modo da visualizzare la finestra di dialogo Project Options; apportare le necessarie modifiche sulla pagina Directories/Conditionals.

Distribuzione dei package

I package vengono distribuiti in modo analogo ad altre applicazioni. I file che vengono distribuiti con un package distribuito possono essere svariati. È necessario distribuire il file bpl e tutti i package o le dll richieste dalla bpl.

La tabella seguente elenca i file che potrebbe essere necesarrio distribuire a seconda dell'uso che si intende fare del package.

Tabella 15.4 File distribuiti con un package

File	Descrizione
ComponentName.h	Consente agli utente di accedere alle interfacce della classe.
ComponentName.cpp	Consente agli utente di accedere al sorgente dell componente.
bpi, obj, and lib	Consente agli utente di eseguire il link dellle applicazioni.

Per informazioni generali sulla distribuzione, fare riferimento al Capitolo 17, "Distribuzione di applicazioni".

Distribuzione di applicazioni che usano package

Quando si distribuisce un'applicazione che usa package di esecuzione, occorre accertarsi che gli utenti abbiano sia il file .exe dell'applicazione sia tutti i file di libreria (.bpl o .dll) chiamati dall'applicazione. Se i file di libreria si trovano in una directory diversa da quella del file .exe, devono essere accessibili tramite il percorso dell'utente. È opportuno seguire la convenzione di mettere i file di libreria nella directory Windows\System. Se si usa InstallShield Express, lo script di installazione può controllare il sistema dell'utente per rilevare quali package sono necessari, prima di reinstallarli alla cieca.

Distribuzione di package ad altri sviluppatori

Se si distribuiscono package di esecuzione o di progettazione ad altri sviluppatori C++Builder, assicurarsi di fornire entrambi i file .bpi e .bpl oltre a tutti gli altri file header necessari. Affinché gli sviluppatori possano collegare staticamente i

componenti nelle proprie applicazioni, vale a dire generare applicazioni che non utilizzano package di esecuzione, sarà necessario distribuire anche i file .lib (o .obj) per tutti i package forniti.

File di raccolta di package

Le raccolte di package (file .dpc) rappresentano un modo pratico per distribuire package ad altri sviluppatori. Ogni raccolta di package contiene uno o più package, inclusi i file bpl e tutti i file aggiuntivi che si desidera distribuire. Quando si seleziona una raccolta di package per installarla nell'IDE, i file costituenti vengono estratti automaticamente dal rispettivo contenitore .pce; la finestra di dialogo Installation offre la possibilità di installare tutti i package inclusi nella raccolta oppure di installarli in modo selettivo.

Per creare una raccolta del package:

1 Scegliere Tools | Package Collection Editor in modo da aprire l'editor Package Collection.

2 Fare clic sul pulsante di scelta rapida Add a Package, quindi selezionare una bpl nella finestra di dialogo Select Package e fare clic su Open. Per aggiungere altre bpl alla raccolta, fare di nuovo clic sul pulsante di scelta rapida Add a Package. Alla sinistra del Package editor appare un diagramma ad albero che visualizza le bpl man mano che vengono aggiunte. Per rimuovere un package, selezionarlo e fare clic sul pulsante di scelta rapida Remove Package.

3 Selezionare il nodo Collection in cima al diagramma ad albero. Sul lato destro dell'editor Package Collection appariranno due campi:

   - Nella casella di testo Author/Vendor Name, è possibile immettere una serie di informazioni facoltative sulla raccolta di package che appariranno nella finestra di dialogo Installation, nel momento in cui gli utenti installeranno i package.
   - Nella casella Directory List, elencare le directory predefinite in cui si vuole che vengano installati i file inclusi nella raccolta di package. Utilizzare i pulsanti Add, Edit e Delete per modificare questo elenco. Per esempio, si supponga che si desideri che tutti i file di codice sorgente vengano copiati nella stessa directory. In questo caso, si dovrebbe immettere il nome `Source` nella casella Directory Name e `C:\MyPackage\Source` nel campo Suggested Path. La finestra di dialogo di Installation visualizzerà `C:\MyPackage\Source` come percorso suggerito per la directory.

4 Oltre alle bpl, la raccolta di package può contenere file .bpi, .obj e .cpp (i file unit), file di documentazione e tutti gli altri file che si desidera includere nella distribuzione. I file subordinati vengono collocati in gruppi di file associati con gli specifici package (bpl); i file in un gruppo vengono installati solo quando viene installata la bpl associata. Per collocare i file subordinati nella raccolta di package, selezionare una bpl nel diagramma ad albero e fare clic sul pulsante di scelta rapida Add File Group; scrivere un nome per il gruppo di file. Se si vuole, aggiungere altri file di gruppo nello stesso modo. Quando si seleziona un gruppo di file, sulla destra del Package Collection editor appariranno dei nuovi campi.

- Dalla casella di riepilogo Install Directory, selezionare la directory in cui si vuole che vengano installati i file del gruppo selezionato. L'elenco a discesa include le directory immesse in Directory List al punto 3 visto in precedenza.
- Selezionare la casella di controllo Optional Group se si vuole che l'installazione dei file nel gruppo selezionato sia facoltativa.
- Nella casella Include Files, elencare i file che si desidera includere nel gruppo selezionato. Utilizzare i pulsanti Add, Delete e Auto per modificare l'elenco. Il pulsante Auto consente di selezionare tutti i file con le estensioni specificate elencati nell'elenco Contains del package; il Package Collection editor utilizza il percorso globale Library Path di C++Builder per cercare questi file.

5 YÈ possibile selezionare le directory di installazione per i package elencati nell'elenco Requires di qualsiasi package incluso nella raccolta. Quando si seleziona una bpl nel diagramma ad albero, sul lato destro del Package Collection editor appaiono quattro nuovi campi:

- Nella casella di riepilogo Required Executables, selezionare la directory in cui si desidera vengano installati i file .bpl dei package elencati nell'elenco Requires. (L'elenco a discesa include le directory immesse in Directory List al punto 3 visto in precedenza). Il Package Collection editor cerca questi file utilizzando il percorso globale Library Path di C++Builder e li elenca in Required Executable Files.
- Nella casella di riepilogo Required Libraries, selezionare la directory in cui si desidera vengano installati i file obj e .bpi dei package inclusi nell'elenco Requires. (L'elenco a discesa include le directory immesse in Directory List al punto 3 visto in precedenza). Il Package Collection editor cerca questi file utilizzando il percorso globale Library Path di C++Builder e li elenca in Required Library Files.

6 Per salvare il file sorgente della raccolta di package, scegliere File|Save. I file sorgenti di una raccolta di package dovrebbero essere salvati con l'estensione .pce.

7 Per generare la raccolta di package, fare clic sul pulsante di scelta rapida Compile. Il Package Collection editor genera un file .dpc avente lo stesso nome del file sorgente (.pce). Se il file sorgente non è ancora stato salvato, prima di compilare l'editor chiede di assegnare un nome al file.

Per modificare o ricompilare un file .pce esistente, selezionare nel Package Collection editor il comando File|Open e individuare il file su cui si vuole operare.

Capitolo

16

Creazione di applicazioni internazionali

In questo capitolo vengono descritte le linee guida per la scrittura di applicazioni che si intende immettere sul mercato internazionale. Pianificando per tempo l'obiettivo, è possibile ridurre la quantità di tempo e di codice necessari per rendere l'applicazione valida anche sui mercati esteri e non solo su quello interno.

Internazionalizzazione e localizzazione

Per creare un'applicazione che possa essere distribuita nei mercati esteri, devono essere effettuate due operazioni fondamentali:

- L'internazionalizzazione
- La localizzazione

Se la versione acquistata di C++Bulider include il modulo Translation Tools, è possibile utilizzarlo per gestire la localizzazione. Per maggiori informazioni, consultare la Guida in linea relativa a Translation Tools (ETM.hlp).

Internazionalizzazione

L'internazionalizzazione è il processo che permette al programma di funzionare in località diverse. Una località è costituita dall'ambiente utente, che comprende le convenzioni culturali del paese di destinazione e la lingua. Windows supporta un ampio numero di località, ciascuna delle quali viene descritta tramite una lingua e una nazione.

Localizzazione

La localizzazione è la procedura di traduzione di un'applicazione in modo che sia in funzione in una determinata località. Oltre alla traduzione delle stringhe che appaiono nell'interfaccia utente, la localizzazione comprende l'adattamento delle funzionalità alle caratteristiche specifiche della nazione. Per esempio, un'applicazione finanziaria può essere modificata in modo da tener conto delle differenti forme di tassazione previste nei diversi stati.

Internazionalizzazione delle applicazioni

Per creare applicazioni internazionalizzate, è necessario attenersi alle seguenti regole generali:

- Abilitare il proprio codice alla gestione di stringhe con set di caratteri internazionali.
- Progettare l'interfaccia utente in modo che possa accettare le modifiche derivanti dalla localizzazione.
- Isolare tutte le risorse che necessitano di una localizzazione.

Abilitazione del codice dell'applicazione

È necessario accertarsi che il codice dell'applicazione sia in grado di gestire le stringhe che incontrerà nelle varie località di destinazione.

Set di caratteri

Le edizioni occidentali di Windows (quella inglese, francese e tedesca incluse) utilizzano il set di caratteri ANSI Latin-1 (1252). Altre edizioni di Windows usano, invece, set differenti di caratteri. Per esempio, la versione giapponese di Windows utilizza il set di caratteri Shift-JIS (code page 932), che rappresenta i caratteri giapponesi come codici di caratteri multibyte.

In generale esistono tre tipi di set di caratteri:

- Singolo byte
- Multibyte
- Caratteri estesi

Windows e Linux supportano entrambi set di caratteri a singolo byte e multibyte oltre a Unicode. In un set di caratteri a singolo byte, ogni byte in una stringa rappresenta un carattere. Il set di caratteri ANSI utilizzato da molti sistemi operativi occidentali è un set di caratteri a singolo byte.

In un set di caratteri multibyte, alcuni caratteri sono rappresentati da un byte e altri da più di un byte. Il primo byte di un carattere multibyte è chiamato il byte iniziale. Di solito, i primi 128 caratteri di un set di caratteri multibyte corrispondono ai caratteri ASCII a 7 bit e tutti i byte il cui valore ordinale è maggiore di 127 è il byte

iniziale di un carattere multibyte. Solo i caratteri a singolo byte possono contenere il valore null (#0). I set di caratteri multibyte – in particolare i set di caratteri a doppio byte (DBCS) – sono ampiamente utilizzati per le lingue asiatiche.

Set di caratteri OEM e ANSI

A volte è necessario effettuare la conversione tra il set di caratteri di Windows (ANSI) e il set di caratteri specificato dalla code page della macchina utente (denominato set di caratteri OEM).

Set di caratteri multibyte

I caratteri ideografici utilizzati in Estremo Oriente non possono usare la semplice mappatura 1:1 tra i caratteri del linguaggio e il tipo *char* a un byte (8 bit). Queste lingue possiedono troppi caratteri per poter essere rappresentati usando il tipo *char* a 1 byte. Invece, una stringa multibyte può contenere uno o più byte per carattere. AnsiStrings può contenere una miscela di caratteri a singolo byte e caratteri multibyte.

Il primo byte di ogni codice di carattere multibyte deriva da un intervallo riservato che dipende dal particolare set di caratteri. Il secondo e i successivi byte possono essere talvolta gli stessi del codice di carattere per un carattere distinto a 1 byte, oppure possono essere compresi nell'intervallo riservato al primo byte dei caratteri multibyte. Pertanto, l'unico modo per distinguere se un determinato byte di una stringa rappresenta un singolo carattere o una parte di un carattere multibyte consiste nel leggere la stringa, partendo dall'inizio, analizzandola in caratteri a due o più byte non appena si incontra un byte iniziale proveniente dall'intervallo riservato.

Quando si scrive un programma per paesi asiatici, bisogna ricordarsi di gestire tutte le operazioni sulle stringhe con funzioni in grado di analizzare le stringhe come caratteri multibyte. Per un elenco delle funzioni RTL che possono operare con caratteri multibyte, consultare le sezioni "International API" e "MCBS utilites" della Guida in linea.

Occorre tenere presente che la lunghezza delle stringhe in byte non corrisponde necessariamente alla lunghezza delle stringhe in caratteri. Si deve fare attenzione a non troncare le stringhe tagliando a metà un carattere multibyte. Non si devono, inoltre, passare caratteri come parametri a funzioni o procedure, poiché le dimensioni di un carattere non possono essere conosciute in prima istanza. Invece, è sempre necessario passare un puntatore a un carattere o a una stringa.

Caratteri estesi

Un altro approccio per lavorare con i set di caratteri ideografici è quello di convertire tutti i caratteri in uno schema di codifica del carattere esteso come Unicode. I caratteri e le stringhe Unicode sono anche chiamati wide characters e wide character string. Nel set di caratteri Unicode, ogni carattere è rappresentato da due byte. Pertanto una stringa Unicode è una sequenza non di singoli byte ma di word composte da due byte.

I primi 256 caratteri Unicode mappano il set di caratteri ANSI. Il sistema operativo Windows supporta Unicode (UCS-2). Il sistema operativo Linux supporta UCS-4, un

superset di UCS-2. C++Builder supporta UCS-2 su entrambe le piattaforme. Poiché i wide character sono lunghi due byte anziché uno, il set di caratteri può rappresentare un numero maggiore di caratteri differenti.

Usando uno schema di codifica dei caratteri esteso, si ha il vantaggio di poter effettuare molte delle solite assunzioni sulle stringhe che non funzionano per i sistemi MBCS. C'è una relazione diretta tra il numero di byte e il numero di caratteri di una stringa. Non occorre preoccuparsi di tagliare i caratteri a metà né di confondere la seconda metà di un carattere con l'inizio di un altro carattere.

Il maggiore inconveniente quando si lavora con i caratteri estesi sta nel fatto che Windows supporta solo poche chiamate alle funzioni API dei caratteri estesi. Per questo motivo, i componenti VCL rappresentano tutti i valori delle stringhe come stringhe a un solo byte o MBCS. La traduzione dal sistema wide character al sistema MBCS tutte le volte che si imposta una proprietà stringa o se ne legge il valore richiede del codice aggiuntivo che rallenta l'applicazione. $Tuttavia, può essere opportuna la traduzione in caratteri estesi per alcuni algoritmi di elaborazione di particolari stringhe che hanno bisogno di sfruttare i vantaggi del mappatura 1:1 tra caratteri e *WideChars*.

Per un elenco delle funzioni RTL che possono operare con caratteri Unicode, consultare la sezione "International API" nella Guida in linea.

Inclusione di funzionalità bidirezionali nelle applicazioni

Alcune lingue non seguono la lettura da sinistra a destra utilizzata solitamente nei paesi occidentali, ma leggono le parole da destra a sinistra e i numeri da sinistra a destra. Tali lingue vengono definite bidirezionali (BiDi) proprio per questa particolarità di comportamento. Le lingue bidirezionali più diffuse sono l'Arabo e l'Ebraico, ma anche altre lingue del Medio-Oriente sono bidirezionali.

TApplication ha due proprietà, *BiDiKeyboard* e *NonBiDiKeyboard*, che consentono di specificare il layout di tastiera. Oltre a ciò, la VCL supporta la localizzazione bidirezionale mediante le proprietà *BiDiMode* e *ParentBiDiMode*. La tabella seguente elenca gli oggetti VCL che hanno queste proprietà:

Tabella 16.1 Oggetti VCL che supportano BiDi

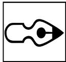
Pagina della Component palette	Oggetto VCL
Standard	TButton
	TCheckBox
	TComboBox
	TEdit
	TGroupBox
	TLabel
	TListBox
	TMainMenu
	TMemo
	TPanel
	TPopupMenu
	TRadioButton

Tabella 16.1 Oggetti VCL che supportano BiDi (continua)

Pagina della Component palette	Oggetto VCL
	TRadioGroup
	TScrollBar
Additional	TActionMainMenuBar
	TActionToolBar
	TBitBtn
	TCheckListBox
	TColorBox
	TDrawGrid
	TLabeledEdit
	TMaskEdit
	TScrollBox
	TSpeedButton
	TStaticLabel
	TStaticText
	TStringGrid
	TValueListEditor
Win32	TComboBoxEx
	TDateTimePicker
	THeaderControl
	THotKey
	TListView
	TMonthCalendar
	TPageControl
	TRichEdit
	TStatusBar
	TTabControl
	TTreeView
Data Controls	TDBCheckBox
	TDBComboBox
	TDBEdit
	TDBGrid
	TDBListBox
	TDBLookupComboBox
	TDBLookupListBox
	TDBMemo
	TDBRadioGroup
	TDBRichEdit
	TDBText
QReport	TQRDBText
	TQRExpr
	TQRLabel

Tabella 16.1 Oggetti VCL che supportano BiDi (continua)

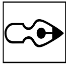
Pagina della Component palette	Oggetto VCL
	TQRMemo
	TQRPreview
	TQRSysData
Altre classi	TApplication (non ha *ParentBiDiMode*)
	TBoundLabel
	TControl (non ha *ParentBiDiMode*)
	TCustomHeaderControl (non ha *ParentBiDiMode*)
	TForm
	TFrame
	THeaderSection
	THintWindow (non ha *ParentBiDiMode*)
	TMenu
	TStatusPanel



THintWindow sceglie il *BiDiMode* del controllo che ha attivato il suggerimento.

Proprietà bidirezionali

Gli oggetti elencati nella Tabella 16.1, "Oggetti VCL che supportano BiDi," a pagina 16-4 hanno le proprietà *BiDiMode* e *ParentBiDiMode*. Queste proprietà, come *BiDiKeyboard* di *TApplication* e *NonBiDiKeyboard*, supportano la localizzazione bidirezionale.



Le proprietà bidirezionali non sono disponibili nella programmazione multipiattaforma.

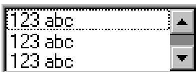
Proprietà BiDiMode

La proprietà *BiDiMode* è un nuovo tipo enumerato, *TBiDiMode*, con quattro stati: *bdLeftToRight*, *bdRightToLeft*, *bdRightToLeftNoAlign* e *bdRightToLeftReadingOnly.*

bdLeftToRight

bdLeftToRight traccia il testo per la lettura da sinistra a destra. L'allineamento e le barre di scorrimento non vengono modificate. Per esempio, quando si immette del testo da destra a sinistra, come nella lingua araba o ebraica, il cursore entra in modalità "spinta" e il testo viene introdotto da destra a sinistra. Il testo nelle lingue neolatine, come l'Inglese o il Francese, viene immesso da sinistra a destra. *bdLeftToRight* è il valore predefinito.

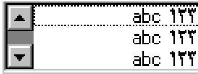
Figura 16.1 TListBox impostato a bdLeftToRight



bdRightToLeft

bdRightToLeft traccia il testo per la lettura da destra a sinistra, l'allineamento viene modificato e la barra di scorrimento viene spostata. Il testo viene immesso nel solito modo per lingue come l'arabo o l'ebraico, che richiedono l'immissione da destra a sinistra. Quando si imposta la tastiera per le lingue neolatine, il cursore entra in modalità "spinta" e il testo viene introdotto da sinistra a destra.

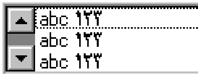
Figura 16.2 TListBox impostato a bdRightToLeft



bdRightToLeftNoAlign

bdRightToLeftNoAlign traccia il testo per la lettura da destra a sinistra, non modifica l'allineamento e non sposta le barre di scorrimento.

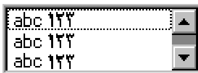
Figura 16.3 TListBox impostato a bdRightToLeftNoAlign



bdRightToLeftReadingOnly

bdRightToLeftReadingOnly riporta il testo da destra a sinistra e non modifica né l'allineamento né le barre di scorrimento.

Figura 16.4 TListBox impostato a bdRightToLeftReadingOnly



Proprietà ParentBiDiMode

ParentBiDiMode è una proprietà booleana. Quando è impostato a **true** (impostazione predefinita), il controllo cerca il proprio genitore per stabilire quale *BiDiMode* usare. Se il controllo è un oggetto *TForm*, la scheda usa l'impostazione di *BiDiMode* di *Application*. Se tutte le proprietà *ParentBiDiMode* sono impostate a **true**, quando si cambia la proprietà *BiDiMode* di *Application*, tutte le schede e i controlli del progetto vengono aggiornati con la nuova impostazione.

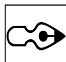
Metodo FlipChildren

Il metodo *FlipChildren* consente di invertire la posizione di un figlio del controllo contenitore. I controlli contenitori sono controlli che accettano altri controlli, come *TForm*, *TPanel* e *TGroupBox*. *FlipChildren* ha un unico parametro booleano, *AllLevels*. Quando è impostato a **false**, solo i figli diretti del controllo contenitore vengono invertiti. Quando è impostato a **true**, vengono invertiti tutti i livelli dei figli del controllo contenitore.

C++Builder inverte i controlli cambiando la proprietà Left e il loro allineamento. Se la parte sinistra di un controllo di testo è a cinque pixel dal margine sinistro del suo controllo genitore, una volta invertito la parte destra del controllo di testo è a cinque

pixel dal margine destro del controllo genitore. Se il controllo di testo è allineato a sinistra, la chiamata di *FlipChildren* farà sì che lo stesso venga allineato a destra.

Per invertire un controllo in fase di progettazione, occorre selezionare Edit|Flip Children, quindi All o Selected, a seconda del fatto che si intendano invertire tutti i controlli o solo i figli del controllo selezionato. È anche possibile invertire un controllo selezionandolo sulla scheda, facendo clic con il pulsante destro e selezionando Flip Children dal menu contestuale.



La selezione di un controllo di testo e l'esecuzione di un comando Flip Children|Selected non produce alcun effetto. Ciò è dovuto al fatto che i controlli di testo non sono contenitori.

Ulteriori metodi

Ci sono diversi altri metodi che possono essere utilizzati per lo sviluppo di applicazioni per utenti di lingue bidirezionali.

Tabella 16.2 Metodi della VCL che supportano BiDi

Metodo	Descrizione
OkToChangeFieldAlignment	Usato con i controlli database. Verifica se l'allineamento di un controllo può essere modificato.
DBUseRightToLeftAlignment	Un wrapper per controlli database destinato alla verifica dell'allineamento.
ChangeBiDiModeAlignment	Cambia il parametro di allineamento che gli viene passato. Non effettua alcuna verifica dell'impostazione di *BiDiMode*, ma si limita a convertire l'allineamento a sinistra nell'allineamento a destra e viceversa, senza modificare i controlli allineati al centro.
IsRightToLeft	Restituisce **true** se è selezionata una delle opzioni da destra a sinistra. Se restituisce **false**, il controllo è in modalità da sinistra a destra.
UseRightToLeftReading	Restituisce **true** se il controllo usa la lettura da destra a sinistra.
UseRightToLeftAlignment	Restituisce **true** se il controllo usa la lettura da destra a sinistra. Può essere ridefinito nel caso occorra personalizzarlo.
UseRightToLeftScrollBar	Restituisce **true** se il controllo usa una barra di scorrimento a sinistra.
DrawTextBiDiModeFlags	Restituisce i flag per tracciare correttamente il testo per il BiDiMode del controllo.
DrawTextBiDiModeFlagsReadingOnly	Restituisce i flag per tracciare correttamente il testo per il BiDiMode del controllo, limitando il flag alla sua modalità di lettura.
AddBiDiModeExStyle	Aggiunge i flag *ExStyle* appropriati al controllo da creare.

Funzioni specifiche della località

È possibile aggiungere alle applicazioni altre funzioni relative a una determinata località. In particolare, per gli ambienti di lingue asiatiche, può essere necessario che

l'applicazione controlli l'editor del metodo di input (IME) che viene usato per convertire la sequenza di tasti premuti dall'utente in stringhe di caratteri.

I componenti della VCL e della CLX offrono il supporto per la programmazione dell'IME. La maggior parte dei controlli con finestra che operano direttamente con l'immissione del testo ha una proprietà *ImeName* che consente di specificare il particolare IME che verrà usato quando il controllo riceve il fuoco di immissione. Essi forniscono anche una proprietà *ImeMode* che specifica come l'IME convertirà l'input da tastiera. *TWinControl* introduce diversi metodi protetti che sono in grado di controllare l'IME dalle classi che vengono definite. Inoltre, la variabile globale *Screen* fornisce informazioni sugli IME disponibili nel sistema dell'utente.

La variabile globale *Screen* (disponibile nella VCL e nella CLX) fornisce anche informazioni sulla mappa di tastiera installata sul sistema dell'utente. È possibile usare questa variabile per ottenere informazioni specifiche della località sull'ambiente in cui viene eseguita l'applicazione.

Progettazione di un'interfaccia utente

Quando si crea un'applicazione per diversi mercati esteri, è importante progettare l'interfaccia utente in modo tale che possa adattarsi alle modifiche necessarie in fase di traduzione.

Testo

Tutto il testo che appare nell'interfaccia utente deve essere tradotto. Il testo in Inglese è quasi sempre più breve di quello tradotto. Gli elementi dell'interfaccia utente devono essere progettati in modo da poter accogliere anche testi più lunghi. Le finestre di dialogo, i menu, le barre di stato e gli altri elementi dell'interfaccia utente che mostrano del testo devono essere creati in modo da poter visualizzare con facilità stringhe più lunghe. Occorre evitare le abbreviazioni—che non esistono nelle lingue che utilizzano caratteri ideografici.

Le stringhe brevi tendono a crescere in fase di traduzione più delle frasi lunghe. La Tabella 16.3 fornisce una stima approssimativa dell'espansione prevista della lunghezza di una stringa in Inglese:

Tabella 16.3 Lunghezza stimata delle stringhe

Lunghezza della stringa in Inglese (in caratteri)	Incremento stimato
1-5	100%
6-12	80%
13-20	60%
21-30	40%
31-50	20%
over 50	10%

Immagini

Teoricamente si possono usare immagini che non richiedono traduzione. Ovviamente, ciò presuppone che esse non contengano testo, il quale richiede sempre una traduzione. Se si deve inserire del testo nelle immagini, è consigliabile utilizzare un oggetto etichetta con uno sfondo trasparente e collocarlo su un'immagine anziché includere il testo come parte dell'immagine.

Ci sono altre considerazioni da fare quando si creano immagini. Bisogna evitare, se possibile, di usare immagini specifiche di una particolare cultura. Per esempio, le caselle di posta assumono un aspetto diverso a seconda dei paesi. I simboli religiosi devono essere evitati se l'applicazione è destinata a paesi che hanno religioni predominanti diverse. Anche i colori possono avere connotazioni simboliche differenti in culture diverse.

Formati e ordinamento

Il formato della data, dell'ora, dei numeri e della valuta nell'applicazione deve essere quello del paese a cui l'applicazione è destinata. Se si utilizzano formati specifici di Windows, non occorre modificarli, in quanto la relativa trasformazione viene effettuata dal registro di Windows dell'utente. Se, invece, si specificano delle stringhe di formato proprie, bisogna ricordarsi di dichiararle come costanti di risorsa in modo che possano essere localizzate.

Anche l'ordine in cui le stringhe vengono ordinate varia da paese a paese. Molte lingue europee includono dei segni diacritici che sono ordinati in modo diverso a seconda del paese. Inoltre, in alcun paesi, le combinazioni di due caratteri sono considerate un singolo carattere in fase di ordinamento. Per esempio, in spagnolo la combinazione *ch* viene ordinata come un'unica lettera intermedia tra la c e la d. A volte un singolo carattere viene ordinato come se fosse costituito da due caratteri separati, come il tedesco *eszett*.

Configurazione della tastiera

Occorre prestare attenzione nell'assegnazione di comandi di scelta rapida ad alcune combinazioni di tasti. Non tutti i caratteri disponibili nelle tastiere americane sono facilmente riproducibili nelle tastiere internazionali. Quando è possibile, per i tasti di scelta rapida è consigliabile utilizzare i tasti numerici e i tasti funzione, in quanto sono disponibili su tutte le tastiere.

Isolamento delle risorse

L'operazione più ovvia nella localizzazione di un'applicazione è la traduzione delle stringhe che appaiono nell'interfaccia utente. Per creare un'applicazione che possa essere tradotta senza dover modificare il codice, bisogna isolare le stringhe dell'interfaccia utente in un unico modulo. C++Builder crea automaticamente un file .dfm (.xfm nelle applicazioni CLX) che contiene le risorse per i menu, le finestre di dialogo e le bitmap.

Oltre a questi elementi ovvi dell'interfaccia utente, occorre isolare anche tutte le stringhe, come i messaggi di errore, che vengono presentate all'utente. Le risorse stringa non vengono incluse nel file scheda, ma è possibile isolarle in un file .RC.

Creazione di DLL di risorse

L'isolamento delle risorse semplifica il processo di traduzione. Il livello successivo di separazione delle risorse è la creazione di DLL di risorse. Una DLL di risorse contiene solo le risorse per un programma. Le DLL di risorse consentono di creare programmi che supportano traduzioni diverse semplicemente cambiando la DLL di risorsa utilizzata.

Per creare DLL di risorse per i propri programmi, si deve ricorrere al Resource DLL wizard. Il Resource DLL wizard richiede un progetto aperto, salvato e compilato. Verrà creato un file RC contenente le tabelle delle stringhe usate dai file RC e le stringhe **resourcestring** del progetto, e verrà generato un progetto per una risorsa di tipo DLL contenente le schede rilevanti e il file RES creato. Quest'ultimo verrà compilato dal nuovo file RC.

Bisogna creare DLL di risorse per ogni traduzione che si intende supportare. Ogni DLL di risorse deve avere una estensione di file specifica del paese a cui è destinata. I primi due caratteri indicano la lingua di destinazione, mentre il terzo carattere indica il paese. Se si usa il Resource DLL wizard, questa operazione viene effettuata automaticamente. Altrimenti, per ottenere il codice locale per la traduzione desiderata, si deve usare il codice seguente:

```
/* This callback fills a listbox with the strings and their associated languages and
countries*/
BOOL __stdcall EnumLocalesProc(char* lpLocaleString)
{
  AnsiString LocaleName, LanguageName, CountryName;
  LCID lcid;
  lcid = StrToInt("$" + AnsiString(lpLocaleString));
  LocaleName = GetLocaleStr(lcid, LOCALE_SABBREVLANGNAME, "");
  LanguageName = GetLocaleStr(lcid, LOCALE_SNATIVELANGNAME, "");
  CountryName = GetLocaleStr(lcid, LOCALE_SNATIVECTRYNAME, "");
  if (lstrlen(LocaleName.c_str()) > 0)
    Form1->ListBox1->Items->Add(LocaleName + ":" + LanguageName + "-" + CountryName);
  return TRUE;
}
/* This call causes the callback to execute for every locale */
  EnumSystemLocales((LOCALE_ENUMPROC)EnumLocalesProc, LCID_SUPPORTED);
```

Uso delle DLL di risorse

L'eseguibile, le DLL e i package (bpls) che costituiscono l'applicazione contengono tutte le risorse necessarie. Tuttavia, per sostituire quelle risorse con le versioni localizzate, è sufficiente distribuire l'applicazione con DLL di risorse localizzate aventi lo stesso nome del file eseguibile, della DLL o del package.

Quando l’applicazione viene avviata, controlla la località del sistema. Se l’applicazione trova delle DLL di risorse con lo stesso nome dei file EXE, DLL o BPL che utilizza, ne controlla l’estensione. Se questa corrisponde a quella della lingua e del paese della località del sistema, l’applicazione utilizza le risorse contenute nella DLL di risorse al posto delle risorse contenute nell’eseguibile, nella DLL o nel package.Se non c’è una DLL di risorse corrispondente alla lingua e al paese, l’applicazione prova a localizzare la DLL di risorse utilizzando quella corrispondente alla lingua della località. Se non ci sono DLL di risorse corrispondenti alla lingua, l’applicazione utilizza le risorse compilate con l’eseguibile, la DLL o il package.

Se si vuole che la propria applicazione utilizzi una DLL di risorse diversa da quelle della località del sistema, si può impostare una voce di ridefinizione della località nel registro di Windows. Sotto HKEY_CURRENT_USER\Software\Borland\Locales key, bisogna aggiungere il percorso e il nome di file della propria applicazione come un valore stringa e impostare il valore dati all’estensione delle proprie DLL di risorse. All’avvio, l’applicazione cercherà le risorse DLL con questa estensione prima di verificare qual è il sistema locale. L’impostazione di questa voce nel registro permette di collaudare le versioni localizzate delle proprie applicazioni senza dover modificare la località del sistema.

Per esempio, la procedura seguente può essere utilizzata in un programma di impostazione o di installazione per definire il valore della chiave del registro che indica la località da utilizzare durante il caricamento delle applicazioni C++Builder:

```
void SetLocalOverrides(char* FileName, char* LocaleOverride)
{
  HKEY Key;
  const char* LocaleOverrideKey = "Software\\Borland\\Locales";
  if (RegOpenKeyEx(HKEY_CURRENT_USER, LocaleOverrideKey, 0, KEY_ALL_ACCESS, &Key)
        == ERROR_SUCCESS) {
    if (lstrlen(LocaleOverride) == 3)
      RegSetValueEx(Key, FileName, 0, REG_SZ, (const BYTE*)LocaleOverride, 4);
    RegCloseKey(Key);
  }
}
```

All’interno dell’applicazione, per ottenere la gestione della DLL di risorse attiva si deve usare la funzione globale *FindResourceHInstance*. Ad esempio:

```
LoadString(FindResourceHInstance(HInstance), IDS_AmountDueName, szQuery, sizeof(szQuery));
```

È possibile distribuire una sola applicazione che si adatti automaticamente alla località del sistema sul quale verrà eseguita, fornendo semplicemente le DLL di risorse appropriate.

Sostituzione dinamica delle DLL di risorse

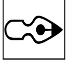
Oltre a localizzare una DLL di risorse in fase di avvio dell’applicazione, è possibile cambiare in modo dinamico le DLL di risorse durante l’esecuzione. Per aggiungere questa funzionalità alle applicazioni, occorre includere nel progetto la unit ReInit. (ReInit è presente nell’esempio Richedit nella directory Examples). Per cambiare lingua, si deve chiamare *LoadResourceModule*, passando la LCID per la nuova lingua, e poi chiamare *ReinitializeForms*.

Per esempio, il codice seguente cambia la lingua dell'interfaccia nel Francese:

```
const  FRENCH = (SUBLANG_FRENCH << 10) | LANG_FRENCH;
if (LoadNewResourceModule(FRENCH))
  ReinitializeForms();
```

Il vantaggio che deriva da questa tecnica è che vengono usate l'istanza attiva dell'applicazione e tutte le sue schede. Non è necessario aggiornare le impostazioni del registro e riavviare l'applicazione o riacquisire le risorse richieste dall'applicazione, come il login ai database server.

Quando si cambia la DLL di risorse, le proprietà specificate nella nuova DLL sovrascrivono quelle delle istanze attiva della scheda.



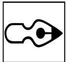
Tutte le modifiche apportate alle proprietà della scheda andranno perdute in fase di esecuzione. Una volta caricata la nuova DLL, non vengono ripristinati i valori predefiniti. Bisogna evitare di scrivere codice che presuma che gli oggetti della scheda vengano reinizializzati al loro stato di avvio, al di là delle differenze dovute alla localizzazione.

Localizzazione delle applicazioni

Una volta che l' applicazione è stata internazionalizzata, si possono creare versioni localizzate per i diversi mercati esteri in cui si intende distribuirla.

Localizzazione delle risorse

In teoria, le risorse vengono isolate in una DLL di risorse contenente i file scheda (.dfm per VCL o .xfm per CLX) e in un file di risorse. È possibile aprire le schede nell'IDE e tradurre le proprietà pertinenti.



In un progetto di DLL di risorse non si possono aggiungere o rimuovere componenti. Tuttavia, è possibile cambiare le proprietà che possono provocare errori di runtime. Per evitare errori, è possibile configurare Object Inspector in modo che visualizzi solo le proprietà che possono essere localizzate; per fare ciò, fare clic destro nell'Object Inspector e utilizzare il menu View per filtrare tutte le categorie di proprietà non desiderate.

È possibile aprire il file RC e tradurre le stringhe rilevanti. A questo scopo, si deve usare lo StringTable editor aprendo il file RC con Project Manager.

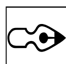
Capitolo

17

Distribuzione di applicazioni

Una volta che l'applicazione C++Builder è pienamente funzionante, la si può distribuire. Cioè, è possibile renderla disponibile ad altri utenti affinché la utilizzino. Per distribuire un'applicazione da utilizzare su altri computer, occorre effettuare una serie di operazioni, in modo che essa risulti completamente funzionale. Il numero di operazioni necessarie varia da un'applicazione all'altra, a seconda del tipo. Le seguenti sezioni trattano alcune problematiche da prendere in esame quando si distribuiscono i vari tipi di applicazioni:

- Distribuzione di applicazioni generiche
- Distribuzione di applicazioni CLX
- Distribuzione di applicazioni database
- Distribuzione di applicazioni Web
- Programmazione per ambienti di destinazione diversi
- Requisiti per la licenza d'uso del software



Le informazioni incluse in queste sezioni valgono per la distribuzione di applicazioni in ambiente Windows.

Distribuzione di applicazioni generiche

Oltre al file eseguibile, un'applicazione può richiedere una certo numero di file di supporto, come DLL, file package e applicazioni di ausilio. Inoltre, il registro di Windows può aver bisogno di contenere dati su un'applicazione, dalla specificazione della dislocazione dei file di supporto alle semplici impostazioni del programma. Il processo di copia dei file di un'applicazione su un computer e di registrazione delle impostazioni necessarie nel registro può essere automatizzato da un programma di installazione, come InstallShield Express. I principali aspetti connessi alla distribuzione dei programmi, comuni a quasi tutti i tipi di applicazione, sono:

- Uso di programmi di installazione
- Identificazione dei file dell'applicazione

Le applicazioni C++Builder che accedono ai database e quelle che funzionano su Web richiedono ulteriori operazioni, oltre a quelle che valgono per le applicazioni generiche. Per ulteriori informazioni sull'installazione di applicazioni database, consultare "Distribuzione di applicazioni database" a pagina 17-6. Per maggiori informazioni sull'installazione di applicazioni Web, consultare "Distribuzione di applicazioni Web" a pagina 17-10. Per maggiori informazioni sull'installazione di controlli ActiveX, consultare "Distribuzione di un controllo ActiveX su Web" a pagina 43-17. Per informazioni sulla distribuzione di applicazioni CORBA, il manuale *VisiBroker Installation and Administration Guide*.

Uso di programmi di installazione

Le applicazioni C++Builder costituite semplicemente da un file eseguibile sono facili da installare su un computer di destinazione. Basta copiare il file eseguibile sul computer. Le applicazioni più complesse, composte da molti file, richiedono, invece, procedure di installazione più articolate. Queste applicazioni richiedono, pertanto, programmi di installazione dedicati.

I kit di installazione automatizzano il processo di creazione dei programmi di installazione, spesso senza che sia necessario scrivere alcuna riga di codice. I programmi creati con questi kit eseguono varie operazioni inerenti all'installazione delle applicazioni C++Builder, fra cui la copia del file eseguibile e dei file di supporto sul computer di destinazione la creazione delle voci nel registro di Windows e l'installazione del Borland Database Engine per le applicazioni database che fanno uso di BDE.

InstallShield Express è un kit di strumenti per l'impostazione che viene abbinato a C++Builder. InstallShield Express è certificato per essere usato con C++Builder e con Borland Database Engine. Si basa sulla tecnologia di Windows Installer (MSI).

InstallShield Express non viene installato automaticamente durante l'installazione di C++Builder, e deve quindi essere installato manualmente se lo si vuole utilizzare per creare programmi di installazione. Per installare InstallShield Express, occorre eseguire il programma di installazione dal CD di C++Builder. Per maggiori informazioni sull'uso di InstallShield Express per creare programmi di installazione, consultare la Guida in linea di InstallShield Express.

Sono disponibili altri kit per l'installazione. Tuttavia, se si distribuisconoo applicazioni database BDE, si dovrebbero usare solo quei kit che si basano sulla tecnologia MSI o quelli certificati per la distribuzione di Borland Database Engine.

Identificazione dei file dell'applicazione

Oltre al file eseguibile, con l'applicazione può essere necessario distribuire un certo numero di altri file.

- File package
- File di applicazione
- Moduli di fusione
- Controlli ActiveX

File di applicazione

Con un'applicazione potrebbe essere necessario distribuire i seguenti tipi di file.

Tabella 17.1 File di applicazione

Tipo	Estensione
File di programma	.exe e .dll
File di applicazione	..bpl, .bpi, e .lib
File della Guida	.hlp, .cnt, e .toc (se utilizzato) o tutti gli altri file di Guida supportati dall'applicazione
ActiveX files	.ocx (a volte supportati da una DLL)
File di tabelle locali	.dbf, .mdx, .dbt, .ndx, .db, .px, .y*, .x*, .mb, .val, .qbe, .gd*

File package

Se l'applicazione usa dei package di runtime, è necessario che essi vengano distribuiti insieme all'applicazione. InstallShield Express gestisce l'installazione dei file package nello stesso modo delle DLL, copiando i file ed apportando le necessarie modifiche nel registro di Windows. Per la distribuzione di package di esecuzione con strumenti di installazione basati su MSI, tra cui InstallShield Express, è possibile anche utilizzare moduli di fusione. Per i dettagli, vedere la sezione successiva.

Borland consiglia di installare i file package di runtime forniti da Borland nella directory Windows\System. Questa directory serve come dislocazione comune, in modo che più applicazioni possano accedere a un'unica istanza dei file. Si consiglia di installare nella stessa directory dell'applicazione anche i package creati. Devono essere distribuiti solo i file .bpl.

Se si distribuiscono package con applicazioni CLX, è necessario includere clx60.bpl invece di vcl60.bpl.

Se i package vengono distribuiti ad altri sviluppatori, fornire i file .bpl, .bpi, e .lib (se si vuole consentire il link statico con i propri package).

Moduli di fusione

InstallShield Express 3.0 si basa sulla tecnologia di Windows Installer (MSI). Ecco perché C++Builder include i moduli di fusione. I moduli di fusione forniscono un metodo standard che è possibile utilizzare per distribuire con le applicazioni codice condiviso, file, risorse, voci del file di Registro e logica di installazione come un singolo file composto. È possibile utilizzare i moduli di fusione per la distribuzione di package di esecuzione con strumenti di installazione basati su MSI tra cui InstallShield Express.

Le librerie di esecuzione contengono alcune interdependenze a causa del modo in cui vengono assemblate. Il risultato è che quando un package viene aggiunto a un progetto di installazione, lo strumento di installazione aggiunge automaticamente o segnala una dipendenza da uno o più packages. Per esempio, se si aggiunge il modulo di fusione VCLInternet a un progetto di installazione, lo strumento di

installazione dovrebbe anche aggiungere automaticamente o segnalare una dipendenza dai moduli VCLDatabase e StandardVCL.

Le dipendenze per ciascun modulo di fusione sono elencate nella tabella seguente. I diversi strumenti di installazione possono reagire a queste dipendenze in modo diverso. InstallShield for Windows Installer aggiunge automaticamente i moduli necessari se è in grado di individuarli. Altri strumenti possono segnalare semplicemente una dipendenza oppure generare un malfunzionamento di creazione se nel progetto non sono inclusi tutti i moduli necessari.

Tabella 17.2 Moduli di fusione e rispettive dipendenze

Modulo di fusione	BPL incluse	Dipendenze
ADO	adortl60.bpl	DatabaseRTL, BaseRTL
BaseClientDataSet	cds60.bpl	DatabaseRTL, BaseRTL, DataSnap, dbExpress
BaseRTL	rtl60.bpl	No dependencies
BaseVCL	vcl60.bpl, vclx60.bpl	BaseRTL
BDEClientDataSet	bdecds60.bpl	BaseClientDataSet, DataBaseRTL, BaseRTL, DataSnap, dbExpress, BDERTL
BDEInternet	inetdbbde60.bpl	Internet, DatabaseRTL, BaseRTL, BDERTL
BDERTL	bdertl60.bpl	DatabaseRTL, BaseRTL
DatabaseRTL	dbrtl60.bpl	BaseRTL
DatabaseVCL	vcldb60.bpl	BaseVCL, DatabaseRTL, BaseRTL
DataSnap	dsnap60.bpl	DatabaseRTL, BaseRTL
DataSnapConnection	dsnapcon60.bpl	DataSnap, DatabaseRTL, BaseRTL
DataSnapCorba	dsnapcrba60.bpl	DataSnapConnection, DataSnap, DatabaseRTL, BaseRTL, BaseVCL
DataSnapEntera	dsnapent60.bpl	DataSnap, DatabaseRTL, BaseRTL, BaseVCL
DBCompatVCL	vcldbx60.bpl	DatabaseVCL, BaseVCL, BaseRTL, DatabaseRTL
dbExpress	dbexpress60.bpl	DatabaseRTL, BaseRTL
dbExpressClientDataSet	dbxcds60.bpl	BaseClientDataSet, DataBaseRTL, BaseRTL, DataSnap, dbExpress
DBXInternet	inetdbxpress60.bpl	Internet, DatabaseRTL, BaseRTL, dbExpress, DatabaseVCL, BaseVCL
DecisionCube	dss60.bpl	TeeChart, BaseVCL, BaseRTL, DatabaseVCL, DatabaseRTL, BDERTL
FastNet	nmfast60.bpl	BaseVCL, BaseRTL
InterbaseVCL	ibxpress60.bpl	BaseClientDataSet, BaseRTL, BaseVCL, DatabaseRTL, DatabaseVCL, DataSnap, dbExpress
Internet	inet60.bpl, inetdb60.bpl	DatabaseRTL, BaseRTL
InternetDirect	indy60.bpl	BaseVCL, BaseRTL
Office2000Components	dcloffice2k60.bpl	DatabaseVCL, BaseVCL, DatabaseRTL, BaseRTL
QuickReport	qrpt60.bpl	BaseVCL, BaseRTL, BDERTL, DatabaseRTL

Tabella 17.2 Moduli di fusione e rispettive dipendenze (continua)

Modulo di fusione	BPL incluse	Dipendenze
SampleVCL	vclsmp60.bpl	BaseVCL, BaseRTL
TeeChart	tee60.bpl, teedb60.bpl, teeqr60.bpl, teeui60.bpl	BaseVCL, BaseRTL
VCLIE	vclie60.bpl	BaseVCL, BaseRTL
VisualCLX	visualclx60.bpl	BaseRTL
VisualDBCLX	visualdbclx60.bpl	BaseRTL, DatabaseRTL, VisualCLX
WebDataSnap	webdsnap60.bpl	XMLRTL, Internet, DataSnapConnection, DataSnap, DatabaseRTL, BaseRTL
WebSnap	websnap61.bpl, vcljpg60.bpl	WebDataSnap, XMLRTL, Internet, DataSnapConnection, DataSnap, DatabaseRTL, BaseRTL, BaseVCL
XMLRTL	xmlrtl60.bpl	Internet, DatabaseRTL, BaseRTL

Controlli ActiveX

Determinati componenti abbinati a C++Builder sono i controlli ActiveX. Il wrapper del componente viene collegato al file eseguibile dell'applicazione (o a un package di runtime), insieme alla quale deve essere distribuito anche il file .ocx per il componente. Questi componenti comprendono:

- Chart FX, copyright SoftwareFX Inc.
- VisualSpeller Control, copyright Visual Components, Inc.
- Formula One (spreadsheet), copyright Visual Components, Inc.
- First Impression (VtChart), copyright Visual Components, Inc.
- Graph Custom Control, copyright Bits Per Second Ltd.

I controlli ActiveX creati dal programmatore devono essere registrati sul computer di distribuzione prima di poter essere usati. Programmi di installazione come InstallShield Express automatizzano questo processo di registrazione. Per registrare manualmente un controllo ActiveX, scegliere il comando Run | ActiveX Server nell'IDE, usare l'applicazione demo TRegSvr oppure il programma di utilità REGSRV32.EXE di Microsoft (non incluso nelle versioni di Windows 9x).

Con un'applicazione devono essere distribuite anche le DLL che supportano i controlli ActiveX.

Applicazioni di ausilio

Le applicazioni di ausilio sono programmi distinti senza i quali l'applicazione C++Builder è parzialmente o completamente impossibilitata a funzionare. Queste applicazioni possono essere quelle fornite col sistema operativo, da Borland oppure prodotti di terze parti. Un esempio di applicazione di ausilio è il programma di utilità Server Manager di InterBase, che amministra i database, gli utenti e la sicurezza di InterBase.

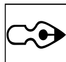
Se un'applicazione dipende da un programma di ausilio, bisogna accertarsi di distribuirlo con l'applicazione, se possibile. La distribuzione di programmi di ausilio possono essere regolata da accordi di licenza per la ridistribuzione. Per informazioni specifiche, consultare la documentazione dell'applicazione di ausilio.

Ubicazione delle DLL

I file DLL usati da una singola applicazione possono essere installati nella sua stessa directory. Le DLL che verranno usate da più applicazioni devono essere installate in una posizione accessibile alle diverse applicazioni. Una convenzione diffusa per la sistemazione di tali DLL comunitarie è quella di collocarle nella directory Windows o Windows\System. Un metodo ancora migliore consiste nel creare un'apposita directory per i file DLL comuni, simile al metodo adottato per l'installazione di Borland Database Engine.

Distribuzione di applicazioni CLX

Per distribuire un'applicazione CLX in ambiente Windows, seguire le stesse procedure usate per la distribuzione delle normali applicazioni. Per informazioni sulla distribuzioe di applicazioni generiche, consultare "Distribuzione di applicazioni generiche" a pagina 17-1. Per informazioni sull'installazione di applicazioni database CLX, consultare "Distribuzione di applicazioni database" a pagina 17-6.



Quando si distribuiscono applicazioni CLX in ambiente Windows, è necessario includere con l'applicazione qtintf.dll in modo da includere il runtime CLX. Se si distribuiscono package con applicazioni CLX, è necessario includere clx60.bpl invece che vcl60.bpl.

Per informazioni sulla scrittura di applicazioni a CLX, consultare il Capitolo 14, "Sviluppo di applicazioni multipiattaforma".

Distribuzione di applicazioni database

Le applicazioni che accedono ai database implicano, al di là della copia del file eseguibile dell'applicazione sul computer host, particolari considerazioni sull'installazione. Spesso l'accesso al database viene gestito da un motore di database distinto, i cui file non possono essere collegati nel file eseguibile dell'applicazione. I file di dati, quando non sono creati in anticipo, devono essere resi disponibili all'applicazione. I database multi-tier richiedono anche una gestione più specialistica dell'installazione, in quanto i file che costituiscono l'applicazione vengono normalmente collocati su più computer.

Poiché sono supportate varie tecnologie di database (ADO, BDE, dbExpress e InterBase Express), i requisiti di distribuzione sono diversi per ognuna di esse. Indipendentemente dalla tecnologia utilizzata, è necessario accertarsi che sul sistema su cui si prevede di eseguire l'applicazione database sia installato il software lato client. ADO, BDE, dbExpress e InterBase Express necessitano anche di driver per interagire con il software lato client del database.

Informazioni specifiche su come distribuire applicazioni database dbExpress, BDE e multi-tier sono riportate nelle seguenti sezioni:

- Distribuzione di applicazioni database dbExpress

- Distribuzione di applicazioni BDE
- Distribuzione di applicazioni database multi-tier (DataSnap)

Le applicazioni database che utilizzano dataset client come *TClientDataSet* o provider di dataset richiedono l'inclusione di midaslib.dcu e di crtl.dcu (per il collegamento statico nel caso si distribuisca un eseguibile indipendente); se si sta creando un package dell'applicazione (con l'eseguibile e tuue le DLL necessarie), si deve includere Midas.dll.

Se si distribuiscono applicazioni database che utilizzano ADO, è necessario accertarsi che sul sistema su cui si prevede di eseguire l'applicazione sia installato MDAC versione 2.1 o successiva. MDAC viene installato automaticamente con software come Windows 2000 o Internet Explorer versione 5 o successiva. È necessario anche assicurarsi che sul client siano installati i driver per il server di database a cui ci si vuole collegare. Non sono necessarie altre operazioni per la distribuzione.

Se si distribuiscono applicazioni database che utilizzano InterBase Express, è necessario accertarsi che sul sistema su cui si prevede di eseguire l'applicazione sia installato il client InterBase. InterBase richiede che gd32.dll e interbase.msg siano situati in una directory accessibile. Non sono necessarie altre operazioni. I componenti InterBase Express comunicano direttamente con l'API di InterBase Client e non richiedono driver aggiuntivi. Per ulteriori informazioni, fare riferimento al documento Embedded Installation Guide reperibile sul sito Web di Borland.

Oltre alle tecnologie qui descritte, per consentire l'accesso ai database da parte delle applicazioni C++Builder è possibile usare motori di database di produttori terzi. Per quanto riguarda i diritti di ridistribuzione, l'installazione e la configurazione, consultare la documentazione fornita col motore di database o il rivenditore.

Distribuzione di applicazioni database dbExpress

dbExpress è un gruppo di driver nativi leggeri che consentono un accesso rapido alle informazioni dei database. dbExpress supporta lo sviluppo multipiattaforma poiché sono disponibili anche su Linux. Per ulteriori informazioni sull'utilizzo dei componenti dbExpress, fare riferimento al Capitolo 26, "Uso di dataset unidirezionali".

Le applicazioni dbExpress possono essere distribuite o come file eseguibile indipendente o come file eseguibile che include le DLL associate dei driver dbExpress.

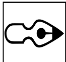
Per distribuire le applicazioni dbExpress come file eseguibili indipendenti, i file oggetto dbExpress devono essere collegati nell'eseguibile in modo statico. L'operazione viene condotta includendo i seguenti file DCU, inclusi nella directory lib:

Tabella 17.3 Distribuzione di dbExpress come eseguibile indipendente

Unit di Database	Da includere in
dbExpINT	Applicazioni che si collegano a database InterBase
dbExpORA	Applicazioni che si collegano a database Oracle

Tabella 17.3 Distribuzione di dbExpress come eseguibile indipendente (continua)

Unit di Database	Da includere in
dbExpDB2	Applicazioni che si collegano a database DB2
dbExpMYS	**Applications connecting to** MySQL 3.22.x
dbExpMYSQL	Applicazioni che si collegano a database MySQL 3.23.x
crtl	richiesto da tutti gli eseguibili che usano dbExpress
MidasLib	Richiesto da eseguibili dbExpress che utilizzano dataset client come *TClientDataSet*



Nel caso di applicazioni database che utilizzano Informix non è possibile distribuire un eseguibile indipendente. Si deve invece distribuire un file eseguibile con la DLL del driver dbexpinf.dll (elencata nella tabella seguente).

Se non si sta distribuendo un eseguibile indipendente, è possibile distribuire assieme all'eseguibile i driver dbExpress associati e le DLL di DataSnap. La seguente tabella elenca le DLL necessarie e quando includerle:

Tabella 17.4 Distribuzione dbExpress con DLL di driver

DLL di database	Da distribuire in
dbexpinf.dll	Applicazioni che si connettono a database Informix
dbexpint.dll	Applicazioni che si connettono a database InterBase
dbexpora.dll	Applicazioni che si connettono a database Oracle
dbexpdb2.dll.	Applicazioni che si connettono a database DB2
dbexpmys.dll	Applicazioni che si connettono a database MySQL 3.22.x
dbexpmysql.dll	Applicazioni che si connettono a database MySQL 3.23.x
Midas.dll	Richiesta da applicazioni database che utilizzano dataset client

Distribuzione di applicazioni BDE

Borland Database Engine (BDE) definisce una vasta API per interagire con i database. Fra tutti i meccanismi per l'accesso ai dati, BDE è quello che supporta la più vasta gama di funzioni e dispone di numerosi con programmi di utilità di supporto. È la migliore soluzione per operare con dati in tabelle Paradox o dBASE.

L'accesso al database per un'applicazione viene fornito da diversi motori di database. Un'applicazione può usare BDE o un motore di database di produttori terzi. Per consentire l'accesso nativo ai sistemi di database SQL vengono forniti gli SQL Links (non disponibili in tutte le edizioni). Le seguenti sezioni descrivono l'installazione degli elementi di accesso al database di un'applicazione:

- Borland Database Engine
- SQL Links

Borland Database Engine

Affinché i componenti dati standard C++Builder abbiano accesso al database, deve essere presente e accessibile Borland Database Engine (BDE). Per i diritti specifici e le limitazioni sulla ridistribuzione di BDE, consultare il documento BDEDEPLOY.

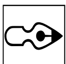
Borland consiglia l'uso di InstallShield Express (o di un altro programma di installazione certificato) per installare BDE. InstallShield Express effettuerà gli inserimenti necessari nel registro e definirà tutti gli alias di cui l'applicazione può avere bisogno. L'uso di un programma di installazione certificato per la distribuzione dei file BDE e dei suoi subset è importante in quanto:

- L'installazione non corretta di BDE o dei suoi subset può provocare errori nell'esecuzione delle applicazioni che usano BDE. Tali applicazioni includono non solo prodotti Borland, ma molti programmi di produttori terzi che usano BDE.
- In ambiente Windows a 32 bit come 95/NT e versioni successive, le informazioni sulla configurazione di BDE sono memorizzate nel registro di Windows anziché nei file .ini, come avveniva in Windows a 16 bit. L'inserimento e la cancellazione dei dati per l'installazione e la disinstallazione sono operazioni complesse.

È possibile installare solo la parte di BDE di cui un'applicazione ha realmente bisogno. Per esempio, se un'applicazione usa solo tabelle Paradox, è necessario installare solo quella parte di BDE necessaria per accedere a tali tabelle. Ciò per un'applicazione vuol dire minore necessità di spazio su disco. I programmi di installazione certificati, come InstallShield Express, sono in grado di eseguire installazioni parziali di BDE. Assicurarsi di lasciare i file BDE di sistema che non vengono usati dall'applicazione distribuita, ma che sono necessari per altri programmi.

SQL Links

SQL Links fornisce i driver che collegano un'applicazione (tramite Borland Database Engine) con il software client per un database SQL. Per i diritti specifici e le limitazioni sulla ridistribuzione di SQL Links, consultare il documento DEPLOY. Come avviene per Borland Database Engine (BDE), SQL Links deve essere distribuito usando InstallShield Express (o un altro programma di installazione certificato).



SQL Links collega solo BDE al software client, non al database SQL stesso. È sempre necessario installare il software client per il sistema di database SQL usato. Per maggiori informazioni sull'installazione e la configurazione del software client, consultare la documentazione del sistema di database SQL o il relativo produttore.

La Tabella 17.5 mostra i nomi del driver e dei file di configurazione utilizzati da SQL Links per la connessione ai diversi sistemi di database SQL. Questi file vengono forniti con SQL Links e sono ridistribuibili secondo quanto previsto dall'accordo di licenza d'uso di C++Builder.

Tabella 17.5 File software del client di database

Produttore	File ridistribuibili
Oracle 7	SQLORA32.DLL and SQL_ORA.CNF
Oracle 8	SQLORA8.DLL and SQL_ORA8.CNF

Tabella 17.5 File software del client di database (continua)

Produttore	File ridistribuibili
Sybase Db-Lib	SQLSYB32.DLL and SQL_SYB.CNF
Sybase Ct-Lib	SQLSSC32.DLL and SQL_SSC.CNF
Microsoft SQL Server	SQLMSS32.DLL and SQL_MSS.CNF
Informix 7	SQLINF32.DLL and SQL_INF.CNF
Informix 9	SQLINF9.DLL and SQL_INF9.CNF
DB/2 2	SQLDB232.DLL and SQL_DB2.CNF
DB/2 5	SQLDB2V5.DLL and SQL_DB2V5.CNF
InterBase	SQLINT32.DLL and SQL_INT.CNF

SQL Links deve essere installato usando InstallShield Express o un altro programma di installazione certificato. Per informazioni specifiche riguardanti l'installazione e la configurazione di SQL Links, consultare il file di Guida SQLLNK32.HLP, installato per impostazione predefinita nella directory principale di BDE.

Distribuzione di applicazioni database multi-tier (DataSnap)

DataSnap fornisce alle applicazioni C++Builder le capacità di database multi-tier, consentendo alle applicazioni client di collegarsi ai provider in un application server.

Installare DataSnap insieme a un'applicazione multi-tier utilizzando InstallShield Express (o un'altro programma di installazione certificato da Borland). Per i dettagli relativi ai file che occorre ridistribuire con un'applicazione, consultare il documento DEPLOY, presente nella directory principale di C++Builder. Consultare inoltre il documento REMOTE per informazioni relative a quali file di DataSnap è possibile ridistribuire e secondo quali modalità.

Distribuzione di applicazioni Web

Alcune applicazioni C++Builder sono progettate per essere eseguite sul World Wide Web, come quelle in formato di DLL Server-side Extension (ISAPI e Apache), le applicazioni CGI e le ActiveForms.

Le operazioni per la distribuzione delle applicazioni Web sono analoghe a quelle previste per le applicazioni generiche, con la differenza che in questo caso i file dell'applicazione devono essere distribuiti sul server Web. Per informazioni sull'installazione delle applicazioni generiche, vedere "Distribuzione di applicazioni generiche" a pagina 17-1. Per informazioni sulla distribuzione di applicazioni database sul Web, consultare "Distribuzione di applicazioni database" a pagina 17-6.

Ecco alcune particolari considerazioni per la distribuzione di applicazioni Web:

- Nel caso di applicazioni database BDE, Borland Database Engine (o un motore di database alternativo) viene installato insieme ai file dell'applicazione sul server Web.

- Nel caso di applicazioni dbExpress, le DLL di dbExpress devono essere incluse nel percorso di ricerca. Nel caso sia incluso, il driver dbExpress viene installato insieme ai file dell'applicazione sul server Web.
- Le politiche di sicurezza per le directory dovrebbero essere impostate in modo che l'applicazione possa accedere a tutti i file di database necessari.
- La directory contenente un'applicazione deve avere attributi di lettura e di esecuzione.
- L'applicazione non deve usare percorsi definiti a livello di codice per l'accesso ai file del database o agli altri file.
- L'ubicazione di un controllo ActiveX viene indicata dal parametro CODEBASE del tag HTML <OBJECT>.

La distribuzione su Apache è trattata nel paragrafo successivo.

Distribuzione su server Apache

WebBroker supporta la versione 1.3.9 e successive di Apache per DLL e applicazioni CGI.

Moduli e applicazioni vengono abilitati e configurati modificando il file httpd.conf di Apache (di solito situato nella directory \conf dell'installazione di Apache).

Abilitazione dei moduli

Le DLL dovrebbero essere ubicate fisicamente nella subdirectory dei moduli di Apache.

Pe abilitare un modulo è necessario apportare due modifiche al file httpd.conf.

1 Aggiungere un elemento LoadModule in modo da consentire ad Apache di individuare e caricare la DLL. Ad esempio:

```
LoadModule MyApache_module modules/Project1.dll
```

2 Aggiungere un elemento resource locator (può essere aggiunto in un punto qualsiasi del file httpd.conf dopo l'elemento LoadModule). Ad esempio:

```
# Sample location specification for a project named project1.

<Location /project1>

   SetHandler project1-handler

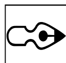
</Location>
```

Questa modifica consente di passare al modulo di Apache tutte le richieste per l'indirizzo http://www.somedomain.com/project1.

La direttiva SetHandler specifica l'applicazione Web server che gestisce la richiesta. L'argomento SetHandler dovrebbe essere impostato al valore della variabile globale ContentType.

Applicazioni CGI

Se si creano applicazioni CGI, la directory fisica (specificata nella direttiva Directory del file httpd.conf) deve avere l'opzione ExecCGI e la clausola SetHandler impostate in modo da consentire l'esecuzione di programmi e poter pertanto eseguire la procedura CGI. Per essere certi che le autorizzazioni siano configurate correttamente, utilizzare la direttiva Alias con entrambe le opzioni ExecCGI e SetHandler abilitate.



Un metodo alternativo consiste nell'utilizzare la direttiva ScriptAlias (senza Options ExecCGI), ma l'utilizzo di questo metodo potrebbe impedire all'applicazione CGI di leggere un qualsiasi file incluso nella directory ScriptAlias.

La seguente riga di httpd.conf è un esempio di come utilizzare la direttiva Alias per creare sul proprio server una directory virtuale e contrassegnare l'esatta ubicazione dello script CGI:

```
Alias/MyWeb/"c:/httpd/docs/MyWeb/"
```

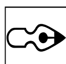
Ciò consente di soddisfare richieste del tipo `/MyWeb/mycgi.exe` eseguendo lo script `c:\httpd\docs\MyWeb\mycgi.exe`.

È possibile anche impostare Options ad All o a ExecCGI utilizzando la direttiva Directory inclusa in httpd.conf. La direttiva Options controlla quali funzioni del server sono disponibili in una particolare directory.

Le direttive Directory sono utilizzate per includere un set di direttive che hanno valore per la directory in oggetto e per le directory in essa contenute. Di seguito e riportato un esempio della direttiva Directory:

```
<Directory "c:/httpd/docs/MyWeb">
    AllowOverride None
    Options ExecCGI
    Order allow,deny
    Allow from all
</Directory>
```

In questo esempio, Options è impostato a ExecCGI, permettendo così l'esecuzione di procedure CGI nella directory MyWeb. La clausola AddHandler consente ad Apache di sapere che i file con estensioni quali as exe e cgi sono script CGI (eseguibili)



Apache viene eseguito in locale sul server, all'interno dell'account specificato nella direttiva User nel file httpd.conf. Accertarsi che l'utente abbia gli opportuni diritti per accedere alle risorse necessarie all'applicazione.

Per ulteriori informazioni sulla distribuzione, consultare il file LICENSE di Apache, incluso nella distribuzione del software. Per ulteriori configurazioni di Apache, consultare il sito Web Apache all'indirizzo www.apache.org.

Programmazione per ambienti di destinazione diversi

Data la natura dei vari ambienti di sistema operativo, vi sono diversi fattori che variano a seconda delle preferenze dell'utente o della configurazione. I seguenti fattori possono avere effetti su un'applicazione distribuita in un altro computer:

Risoluzione dello schermo e numero di colori

Le dimensioni del desktop e il numero di colori disponibili su un computer sono configurabili e dipendono dall'hardware installato. Questi attributi saranno probabilmente diversi a seconda che si tratti dei computer di distribuzione o di quelli di sviluppo.

L'aspetto di un'applicazione (finestra, oggetto e dimensioni del font) su computer configurati per risoluzioni di schermo differenti può essere gestito in vari modi:

- Progettare l'applicazione per la risoluzione più bassa prevista (normalmente, 640x480). Non effettuare particolari operazioni per ridimensionare dinamicamente gli oggetti in modo da renderli proporzionali al display dello schermo del computer host. Visivamente gli oggetti appariranno tanto più piccoli quanto più alta sarà la risoluzione impostata.
- Progettare adottando qualsiasi risoluzione dello schermo sul computer di sviluppo e, in fase di esecuzione, ridimensionare dinamicamente tutte le schede e gli oggetti in modo proporzionale alla differenza tra la risoluzione dello schermo dei computer di sviluppo e quella dei computer di distribuzione (un rapporto della differenza di risoluzione dello schermo).
- Progettare utilizzando qualsiasi risoluzione dello schermo sul computer di sviluppo e, in fase di esecuzione, ridimensionate dinamicamente solo le schede dell'applicazione. A seconda della dislocazione dei controlli visuali sulle schede, ciò può richiedere che possa essere effettuato lo scorrimento delle schede perché l'utente sia in grado di accedere a tutti i relativi controlli.

Considerazioni sul ridimensionamento non dinamico

Se le schede e i controlli visuali che costituiscono un'applicazione non vengono ridimensionati dinamicamente in fase di esecuzione, occorre progettare gli aspetti che costituiscono l'applicazione per la risoluzione più bassa. Altrimenti, le schede di un'applicazione che funzionano su un computer configurato per una risoluzione dello schermo più bassa di quella del computer di sviluppo potrebbe oltrepassare i margini dello schermo.

Per esempio, se il computer di sviluppo è impostato per una risoluzione di 1024x768 e una scheda viene progettata con una larghezza di 700 pixel, la scheda non risulterà del tutto visibile all'interno del desktop su un computer configurato per una risoluzione di 640x480.

Considerazioni sul ridimensionamento dinamico di schede e controlli

Se le schede e i controlli visuali di un'applicazione vengono ridimensionati in modo dinamico, bisogna adattare tutte le fasi del processo di ridimensionamento per

assicurare un aspetto ottimale dell'applicazione sotto tutte le possibili risoluzioni dello schermo. Quando si ridimensionano dinamicamente gli elementi visuali di un'applicazione, occorre considerare i seguenti fattori:

- Il ridimensionamento delle schede e dei controlli visuali viene effettuato con un rapporto calcolato confrontando la risoluzione dello schermo del computer di sviluppo con quella del computer su cui l'applicazione viene installata. Usare una costante per rappresentare una dimensione della risoluzione dello schermo sul computer di sviluppo: l'altezza o la larghezza, in pixel. Ricavare la stessa dimensione per il computer dell'utente in fase di esecuzione usando la proprietà *TScreen::Height* o *TScreen::Width* . Dividere il valore ottenuto per il computer di sviluppo per il valore del computer dell'utente; in questo modo, si otterrà il rapporto della differenza tra le risoluzioni dello schermo dei due computer.
- Ridimensionare gli elementi visuali dell'applicazione (schede e controlli) riducendo o aumentando le dimensioni degli elementi e la loro posizione sulle schede. Questo ridimensionamento risulta proporzionale alla differenza tra la risoluzione dello schermo sul computer di sviluppo e quella del computer dell'utente. Ridimensionare e riposizionare automaticamente i controlli visuali sulle schede impostando la proprietà *TCustomForm::Scaled* a **true** e chiamando il metodo *TWinControl::ScaleBy* (oppure*TWidgetControl.ScaleBy* nel caso di applicazioni mutipiattaforma). Il metodo *ScaleBy*, comunque, non modifica l'altezza e la larghezza della scheda. Per eseguire questa operazione manualmente, basta moltiplicare i valori attivi delle proprietà *Height* e *Width* per il rapporto della differenza tra le risoluzioni degli schermi.
- I controlli su una scheda possono essere ridimensionati manualmente, anziché automaticamente, con il metodo *TWinControl::ScaleBy* (oppure *TWidgetControl.ScaleBy* nel caso di applicazioni multipiattaforma), referenziando ciascun controllo visuale in un ciclo e impostandone dimensioni e posizione. I valori delle proprietà *Height* e *Width* dei controlli visuali vanno moltiplicati per il rapporto della differenza tra le risoluzioni degli schermi. Riposizionare i controlli visuali in modo proporzionale alla differenza delle risoluzione degli schermi moltiplicando i valori delle proprietà *Top* e *Left* per lo stesso rapporto.
- Se un'applicazione viene progettata su un computer configurato per una risoluzione più alta di quella del computer dell'utente, le dimensioni dei font saranno ridotte proporzionalmente nel processo di ridimensionamento dei controlli visuali. Se la dimensione del font in fase di progettazione è troppo piccola, il font ridimensionato in fase di esecuzione potrebbe risultare non leggibile. Ad esempio, la dimensione predefinita del font di una scheda è 8. Se il computer di sviluppo ha una risoluzione pari a 1024x768 e quello dell'utente una risoluzione pari a 640x480, le dimensioni del controllo visuale verranno ridotte di un fattore 0,625 (640 / 1024 = 0,625). La dimensione originale del font di 8 verrà ridotta a 5 (8 * 0,625 = 5). Il testo nell'applicazione apparirà irregolare e illeggibile quando verrà visualizzato con un font ridotto.
- Alcuni controlli visuali, come *TLabel* e *TEdit*, vengono ridimensionati dinamicamente quando si modificano le dimensioni del font per il controllo. Ciò può avere effetti sulle applicazioni distribuite quando le schede e i controlli vengono ridimensionati in modo dinamico. Il ridimensionamento del controllo dovuto alla modifica delle dimensioni del font viene sommato alla modifica delle

dimensioni dovuta al ridimensionamento proporzionale per le risoluzioni degli schermi. Questo effetto viene compensato impostando la proprietà *AutoSize* di questi controlli a **false**.

- Evitare l'uso esplicito delle coordinate in pixel, come quando si disegna direttamente su un canvas. Modificare, invece, le coordinate per un rapporto proporzionato alla differenza delle risoluzioni degli schermi tra il computer di sviluppo e il computer dell'utente. Per esempio, se l'applicazione disegna su un canvas un rettangolo alto dieci pixel e largo venti, basta moltiplicare tali numeri per il rapporto della differenza delle risoluzioni degli schermi. Questa operazione garantisce che il rettangolo apparirà con le stesse dimensioni con risoluzioni dello schermo differenti.

Regolazione in funzione del numero di colori disponibili

Se si vuole tenere conto del fatto che tutti i computer di distribuzione non sono configurati con la stessa disponibilità di colori, il modo più sicuro per aver buoni risultati è quello di usare grafici con il minimo numero possibile di colori. Ciò è particolarmente vero per i glifi dei controlli, che normalmente usano della grafica a 16 colori. Per la visualizzazione delle figure, o vengono diverse fornite copie delle immagini con risoluzioni e intensità di colore differenti, oppure vengono indicati i requisiti minimi di risoluzione e di colore per l'applicazione.

Font

Windows viene distribuito con un set standard di font TrueType e vettoriali. Linux viene distribuito con un set standard di font, che variano a seconda della distribuzione. Quando si progetta un'applicazione che deve essere distribuita su altri computer, bisogna tener conto del fatto che non tutti i computer potrebbero avere altri font oltre a quelli del set standard.

Tutti i componenti di testo usati nell'applicazione devono usare i font che sono probabilmente disponibili su tutti i computer di distribuzione.

Quando in un'applicazione è assolutamente necessario usare un font non standard, occorre distribuire tale font con l'applicazione. Oppure, si deve fare in modo che il programma di installazione o l'applicazione stessa installino il font sul computer di distribuzione. La distribuzione dei font di produttori terzi può essere soggetta a limitazioni imposte da chi ha creato il font.

Windows prevede una misura di sicurezza che consente di usare un font che sul computer non esiste. Lo sostituisce con un altro font esistente che ritiene essere quello più simile. Anche se è vero che ciò può impedire l'insorgere di errori relativi alla mancanza di font, il risultato finale può essere un degrado dell'aspetto visuale dell'applicazione. È meglio prepararsi a questa eventualità in fase di progettazione.

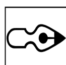
Per rendere disponibile in un'applicazione Windows un font non standard, usare le funzioni API di Windows *AddFontResource* e *DeleteFontResource*. Distribuire il file .fot del font non standard con l'applicazione.

Versioni del sistema operativo

Quando si usano API del sistema operativo o si accede dall'applicazione ad aree del sistema operativo, c'è la possibilità che la funzione, l'operazione o l'area non siano disponibili su computer con versioni diverse del sistema operativo.

Per tener conto di questa possibilità, sono disponibili alcune opzioni:

- Specificare nei requisiti di sistema dell'applicazione le versioni del sistema operativo su cui il programma può funzionare. La responsabilità di installare e di utilizzare l'applicazione solo su versioni di sistema operativo compatibili è a carico dell'utente.
- Controllare la versione del sistema operativo mentre si installa l'applicazione. Se è presente una versione di sistema operativo non compatibile, si deve interrompere la procedura di installazione o quanto meno informare l'utente del problema.
- Controllare in esecuzione la versione del sistema operativo, immediatamente prima di eseguire un'operazione non valida per tutte le versioni. Se è presente una versione di sistema operativo non compatibile, annullare la procedura e notificarlo all'utente. In alternativa, includere sezioni di codice differente da eseguire a seconda della versione del sistema operativo.



Alcune operazioni vengono eseguite in Windows 95/98 in modo differente rispetto a Windows NT/2000/XP. Per determinare la versione in uso di Windows, ricorrere alla funzione API di Windows *GetVersionEx*.

Requisiti per la licenza d'uso del software

La distribuzione di alcuni file associati alle applicazioni C++Builder è soggetta a limitazioni o non è affatto consentita. I documenti seguenti descrivono gli aspetti legali relativi alla distribuzione di questi file:

- DEPLOY
- README
- Accordo di licenza
- Documentazione sui prodotti di terze parti

DEPLOY

Il documento DEPLOY tratta alcuni aspetti legali connessi alla distribuzione dei diversi componenti, dei programmi di utilità e di altre aree del prodotto che possono far parte o essere associati a un'applicazione C++Builder. Il documento DEPLOY è installato nella directory principale di C++Builder. Gli argomenti trattati comprendono, tra l'altro

- .exe, .dll, e .bpl files
- I componenti e i package di progettazione

- Borland Database Engine (BDE)
- I controlli ActiveX
- Immagini di esempio
- SQL Links

README

Il documento README contiene le informazioni dell'ultimo minuto su C++Builder, oltre a informazioni che potrebbero influenzare i diritti di ridistribuzione dei componenti, dei programmi di utilità o di altre aree del prodotto. Il documento README è installato nella directory principale di C++Builder.

Accordo di licenza

L'accordo di licenza di C++Builder, un documento stampato, tratta gli altri diritti legali e gli obblighi riguardanti C++Builder.

Documentazione sui prodotti di terze parti

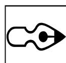
I diritti di ridistribuzione per i componenti, i programmi di utilità, le applicazioni aggiuntive, i motori di database e gli altri prodotti di terzi parti sono stabiliti dall'organismo che fornisce il prodotto. Per informazioni riguardanti la ridistribuzione di tali prodotti con le applicazioni C++Builder, consultare la documentazione relativa al prodotto o al produttore prima di effettuarne la distribuzione.

Parte

II

Sviluppo di applicazioni database

I capitoli inclusi nella sezione "Sviluppo di applicazioni database" presentano i concetti e le tecniche necessarie per creare applicazioni database in C++Builder.



Il livello di supporto per la costruzione di applicazioni database varia a seconda della versione di C++Builder acquistata. In particolare, si deve essere in possesso almeno dell'edizione Professional per utilizzare dataset client e della versione Enterprise per sviluppare applicazioni database multi-tier.

Capitolo

18

Progettazione di applicazioni database

Le applicazioni database consentono agli utenti di interagire con le informazioni memorizzate nei database. Questi forniscono la struttura per le informazioni, e ne consentono la condivisione tra applicazioni differenti.

C++Builder fornisce il supporto per le applicazioni database relazionali. I database relazionali organizzano le informazioni in tabelle, che contengono righe (record) e colonne (campi). Queste tabelle possono essere manipolate attraverso semplici operazioni, note come calcoli relazionali.

Nella progettazione di un'applicazione database occorre capire come sono strutturati i dati. In base alla struttura è possibile, quindi, progettare un'interfaccia per visualizzare i dati all'utente e consentirgli di immettere nuove informazioni o di modificare i dati esistenti.

In questo capitolo vengono riportate alcune considerazioni di carattere generale per la progettazione di un'applicazione database e vengono analizzate le decisioni da adottare per la definizione di un'interfaccia utente.

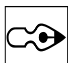
Uso dei databases

C++Builder include molti componenti per accedere ai database e rappresentare le informazioni in essi contenute. Essi sono raggruppati il base al meccanismo di accesso di dati:

- La pagina BDE della Component palette contiene i componenti che utilizzano Borland Database Engine (BDE). Borland Database Engine (BDE) definisce una vasta API per l'interazione con i database. Fra tutti i meccanismi per l'accesso ai dati, BDE è quello che supporta la più vasta gamma di funzioni e dispone del maggior numero di programmi di utilità di supporto. È senza dubbio il miglior modo per operare con dati in tabelle Paradox o dBASE. Tuttavia, è anche il

meccanismo più complicato da distribuire. Per ulteriori informazioni sull'uso dei componenti BDE, consultare il Capitolo 24, "Uso di Borland Database Engine".

- La pagina ADO della Componet palette contiene componenti che utilizzano ActiveX Data Objects (ADO) per accedere tramite OLEDB alle informazioni nel database. ADO è uno standard di Microsoft. Esiste una vasta gamma di driver ADO disponibili per il collegamento a vari server di database. L'utilizzo di componenti basati su ADO permette di integrare l'applicazione in un ambiente basato su ADO (ad esempio, che utilizza application server basati su ADO). Per ulteriori informazioni sull'uso dei componenti ADO, consultare il Capitolo 25, "Operazioni con componenti ADO".
- La pagina dbExpress della Componet palette contiene componenti che utilizzano dbExpress per accedere alle informazioni nel database. dbExpress è un gruppo di driver leggeri che consentono un accesso rapido alle informazioni dei database. Inoltre, i componenti dbExpress supportano lo sviluppo multipiattaforma perché sono disponibili anche in ambiente Linux. Tuttavia, i componenti database dbExpress supportano anche una gamma molto ristretta di funzioni per la manipolazione dei dati. Per ulteriori informazioni sull'uso dei componenti dbExpress, consultare il Capitolo 26, "Uso di dataset unidirezionali".
- La pagina InterBase della Component palette contiene componenti che accedono a database InterBase in modo diretto, senza passare attraverso uno strato di motore separato.
- La pagina Data Access della Component palette contiene componenti che possono essere utilizzati con qualsiasi meccanismo di accesso ai dati. Questa pagina include *TClientDataset*, che può funzionare con dati memorizzati su disco oppure, utilizzando il componente *TDataSetProvider*, anch'esso su questa pagina, con componenti di uno degli altri gruppi. Per ulteriori informazioni sull'uso dei dataset client, consultare il Capitolo 27, "Utilizzo dei dataset client". per ulteriori information su *TDataSetProvider*, vedere il Capitolo 28, "Uso di componenti provider".



Le varie versioni di C++Builder includono driver differenti per l'accesso a server di database utilizzando BDE, ADO o *dbExpress*.

Nella progettazione di un'applicazione database occorre decidere quale set di componenti utilizzare. Ogni meccanismo di accesso di dati è differente per la gamma di funzioni disponibili, per la semplicità di distribuzione e per la disponibilità di driver per supportare vari server di database.

Oltre a scegliere un meccanismo di accesso ai dati, è necessario scegliere un server di database. Esistono vari tipi di database e sarà opportuno soppesare i vantaggi e gli svantaggi di ognuno prima di scegliere un particolare server di database.

Tutti i tipi di database contengono tabelle che memorizzano le informazioni. Inoltre, la maggior parte dei server (ma non tutti) supportano funzioni aggiuntive come

- Sicurezza del database
- Transazioni
- Integrità referenziale, procedure registrate e trigger

Tipi di database

I server di database relazionali agiscono in modo differente a seconda del modo utilizzato per memorizzare le informazioni e a seconda del modo utilizzato per permettere a più utenti di accedere contemporaneamente a quelle informazioni. C++Builder fornisce il supporto per due tipi di server database relazionale:

- I **server di database remoti** di solito risiedono in una macchina separata. A volte, i dati provenienti da un server di database remoto non risiedere nemmeno in una singola macchina, ma sono distribuiti su numerosi server. Benché i server di database remoti si differenzino a seconda del modo utilizzato per memorizzare le informazioni, essi presentano ai client un'interfaccia logica comune. Questa interfaccia comune è SQL (Structured Query Language).Poiché si accede ad essi utilizzando il linguaggio SQL, a volte vengono anche chiamati server SQL. (Un altro nome utilizzato è Remote *Database* Management system o RDBMS.) Oltre ai comuni comandi che compongono il linguaggio SQL, la maggior parte dei server di database remoti supporta un "dialetto" univoco di SQL. Esempi di server SQL sono InterBase, Oracle, Sybase, Informix, Microsoft SQL server e DB2.
- I **database locali** risiedono nell'unità locale o in una rete locale. Essi possiedono API proprietarie per accedere ai dati. Quando sono condivisi da più utenti, usano meccanismi di bloccaggio basati su file di database. A causa di ciò, a volte vengono chiamati database basati su file. Esempi di database locali sono Paradox, dBASE, FoxPro e Access.

Le applicazioni che usano i database locali sono chiamate **applicazioni single-tier** perché l'applicazione e il database condividono un singolo file system. Le applicazioni che utilizzano i server di database remoti sono definite **applicazioni two-tier** o **applicazioni multi-tier** perché l'applicazione e il database operano su sistemi (o tier) indipendenti).

La scelta del tipo di database da utilizzare dipende da numerosi fattori. Ad esempio, i dati possono essere già memorizzati in un database esistente. Se si creano le tabelle di database che saranno utilizzate dall'applicazione, sarà bene porsi le seguenti domande:

- Quanti utenti condivideranno queste tabelle? I server di database remoti sono progettati perché numerosi utenti possano accedervi contemporaneamente. Essi forniscono il supporto per più utenti attraverso un meccanismo chiamato transazioni. Anche alcuni database locali (come Local InterBase) forniscono il supporto per le transazione, mentre la maggior parte fornisce solo meccanismi basati sul blocco dei file e alcuni (come file dei dataset client) non forniscono affatto alcun supporto multiutente.
- Quanti dati conterranno le tabelle? I server di database remoti possono contenere molti più dati dei database locali. Alcuni server di database remoti sono progettati per immagazzinare grandi quantità di dati mentre altri sono ottimizzati per altri criteri (come gli aggiornamenti veloci).
- Quale tipo di prestazioni (velocità) si richiedono al database? Di solito i database locali sono più rapidi dei server di database remoti perché risiedono nello stesso sistema dell'applicazione database. Server di database remoti differenti sono

ottimizzati per supportare tipi di operazioni differenti, pertanto, quando si sceglie un server di database remoto, è bene prestare attenzione alle prestazioni.

- Quale tipo di supporto sarà disponibile per la gestione del database? I database locali richiedono meno supporto dei server di database remoti. Di solito, sono anche meno costosi da far funzionare perché non richiedono server installati separatamente o costose licenze d'uso per ogni installazione.

Sicurezza del database

Spesso i database contengono informazioni riservate. Diversi database forniscono piani di sicurezza per la protezione di queste informazioni. Alcuni, come Paradox e dBASE, forniscono solo sicurezza a livello di tabella o di campo. Quando gli utenti cercano di accedere alle tabelle protette, viene loro richiesto di fornire una password. Una volta che sono stati identificati, gli utenti possono visualizzare solo i campi (o le colonne) ai quali hanno l'accesso.

Molti server SQL richiedono, per poter essere usati, almeno una password e un nome utente. Dopo che l'utente ha effettuato il login al database, il suo nome e la sua password determinano quali tabelle possono essere usate. Per informazioni su come fornire le password ai server SQL, consultare "Controllo del login al server" a pagina 21-4.

Quando si progettano applicazioni database, occorre tener conto del tipo di autentica richiesta dal database server. Spesso, le applicazioni sono progettate per nascondere agli utenti il login esplicito al database, e richiedono che l'utente effettui solo il login all'applicazione. Se non si vuole che gli utenti siano obbligati a fornire una password, occorre usare un database che non la richiede o fornire la password e il nome utente al server da programma. In questo secondo caso si deve prestare attenzione al rischio di aprire una breccia nella sicurezza leggendo la password dall'applicazione.

Se si richiede all'utente di fornire una password, occorre tener conto del momento in cui questa viene richiesta. Se si sta utilizzando un database locale ma si prevede in seguito di passare a un server SQL più grande, potrebbe essere opportuno richiedere la password nel momento in cui si effettua il login al database SQL invece che all'apertura delle singole tabelle.

Se l'applicazione richiede più password perché occorre effettuare il login a diversi sistemi o database protetti, si potrebbe prevedere che gli utenti forniscono un'unica password principale da utilizzare per accedere a una tabella contenente tutte le password richieste dai sistemi protetti. L'applicazione fornisce, quindi, le password da programma, senza richiedere all'utente di fornirne altre.

Nelle applicazioni multi-tier può essere opportuno usare un modello di sicurezza completamente diverso. Si può usare HTTP o COM per controllare l'accesso ai tier intermedi, e lasciare che siano essi a gestire tutti i dettagli del login ai database server.

Transazioni

Una transazione è un gruppo di azioni che devono essere tutte portate a termine con successo su una o più tabelle di un database prima che vengano inoltrate (rese permanenti). Se una delle azioni del gruppo non riesce, tutte le azioni vengono respinte (annullate).

Le transazioni assicurano che

- Tutti gli aggiornamenti di una singola transazione saranno effettuati oppure annullati e riportati al loro stato precedente. Questo comportamento viene definito **atomicità**.
- Una transazione è una trasformazione corretta dello stato di sistema, che preserva gli invarianti di stato. Questo comportamento viene definito **congruenza**.
- Le transazioni simultanee non vedono i rispettivi risultati parziali o non completati, che potrebbero creare inconsistenze nello stato di applicazione. Questo comportamento viene definito **isolamento**.
- Gli aggiornamenti sottoposti ai record sopravvivono ai malfunzionamenti, compresi i malfunzionamenti dovuti a problemi di comunicazione, di processo e del sistema server. Questo comportamento viene definito **resistenza**.

Le transazioni proteggono dai guasti all'hardware che si verificano nel mezzo di un comando o di un set di comandi del database. La registrazione transazionale consente di recuperare lo stato resistente dopo un malfunzionamento dei supporti di memorizzazione su disco. Le transazioni costituiscono anche la base del controllo di simultaneità multiutente sui server SQL. Quando un utente interagisce con il database solo tramite le transazioni, i suoi comandi non possono rompere l'unità della transazione di un altro utente. Invece, il server SQL programma le transazioni in arrivo, che hanno successo o falliscono globalmente.

Il supporto delle transazioni non è incluso nella maggior parte dei database locali, anche se è previsto da Local InterBase. Inoltre, i driver BDE forniscono un supporto transazionale limitato per alcuni database locali. Il supporto delle transazioni sul database viene fornito dal componente che rappresenta la connessione al database. Per dettagli sulla gestione delle transazioni utilizzando un componente database connection, vedere "Gestione delle transazioni" a pagina 21-6.

Nelle applicazioni multi-tiered è possibile creare transazioni che includono azioni diverse dalle operazioni del database o che riguardano più database. Per informazioni dettagliate sull'uso delle transazioni nelle applicazioni multi-tiered, consultare "Gestione delle transazioni nelle applicazioni multi-tiered" a pagina 29-18.

Integrità referenziale, procedure registrate e trigger

Tutti i database relazionali hanno alcune caratteristiche comuni che consentono alle applicazioni di memorizzare e di manipolare i dati. Inoltre, spesso i database forniscono altre caratteristiche specifiche, che possono risultare utili per garantire relazioni coerenti tra le tabelle di un database. Ne fanno parte

- **Integrità referenziale**. L'integrità referenziale fornisce un meccanismo per impedire che le relazioni master/detail tra le tabelle vengano danneggiate. Quando l'utente cerca di cancellare un campo di una tabella master, il che genererebbe record detail orfani, le regole dell'integrità referenziale impediscono la cancellazione o cancellano automaticamente i record detail orfani.
- **Procedure registrate**. Le procedure registrate sono set di istruzioni SQL a cui viene assegnato un nome e che vengono memorizzate su un server SQL. Di solito, queste procedure eseguono sul server normali operazioni relative ai database, e restituiscono set di record (dataset).
- **Trigger**. I trigger sono set di istruzioni SQL che vengono eseguiti automaticamente in risposta a un particolare comando.

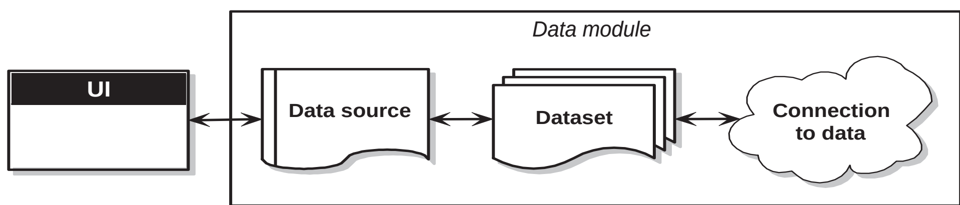
Architettura di un database

Le applicazioni database sono costituite dagli elementi dell'interfaccia utente, dai componenti che rappresentano i dati contenuti dalle tabelle in quei database (dataset). Il modo di organizzare queste parti genera l'architettura dell'applicazione database.

Struttura generale

Benché esistano molti modi differenti per organizzare i componenti in un'applicazione database, la maggior parte segue lo schema generale illustrato nella Figura 18.1:

Figura 18.1 Architettura generale di un database



La scheda dell'interfaccia utente

È buona norma isolare l'interfaccia utente su una scheda che sia completamente separata dal resto dell'applicazione. Questo comporta numerosi vantaggi. Isolando l'interfaccia utente dai componenti che rappresentano le informazioni di database, si introduce una maggior flessibilità nel proprio progetto. Le modifiche al modo in cui vengono gestite le informazioni di database non comporteranno la riscrittura dell'interfaccia utente e le modifiche all'interfaccia utente non comporteranno la modifica di quelle parti dell'applicazione che operano sul database. Inoltre, questo tipo di isolamento permette di sviluppare schede comuni che è possibile condividere tra più applicazioni, fornendo così un'interfaccia utente coerente. Memorizzando collegamenti a schede ben progettate nell'Object Repository, qualunque sviluppatore

può basarsi su blocchi fondamentali già esistenti anziché iniziare ex novo ogni nuovo progetto. La condivisione di schede rende anche possibile sviluppare standard aziendali per le interfacce delle applicazioni. Per ulteriori informazioni sulla creazione dell'interfaccia utente di un'applicazione database, consultare "Progettazione dell'interfaccia utente" a pagina 18-16.

Il modulo dati

Se l'interfaccia utente è stata isolata in schede indipendenti, è possibile utilizzare un modulo dati per alloggiare i componenti che rappresentano le informazioni di database (dataset) e i componenti che connettono questi dataset alle altre parti dell'applicazione. Analogamente alle schede dell'interfaccia utente, è possibile condividere i moduli dati nell'Object Repository in modo che possano essere riutilizzati o condivisi tra le applicazioni.

Il datasource

Il primo elemento nel modulo dati è un datasource. Il datasource agisce come un condotto tra l'interfaccia utente e un dataset che rappresenta le informazioni contenute in un database. Numerosi controlli data-aware su una scheda possono condividere un singolo datasource, nel quale caso ciò che viene visualizzato in ogni controllo è sincronizzato in modo che, man mano che l'utente fa scorrere i record, in ogni controllo viene visualizzato il valore corrispondente presente nei campi del record corrente.

Il dataset

Il cuore dell'applicazione database è il dataset. Questo componente rappresenta un insieme di record provenienti dal database sottostante. Questi record possono essere i dati di una singola tabella di database, un sottoinsieme dei campi o dei record in una tabella oppure informazioni provenienti da più tabelle unite in una singola vista. Grazie all'utilizzo dei dataset, la logica dell'applicazione è esentata dalla necessità di ristrutturare le tabelle fisiche nei database. Nel caso si cambi il database sottostante, potrebbe essere necessario modificare il modo in cui il componente dataset specifica i dati che contiene, ma il resto dell'applicazione potrebbe continuare a funzionare senza modifiche. Per ulteriori informazioni sulle proprietà e sui metodi comuni dei dataset, consultare il Capitolo 22, "I dataset".

La connessione ai dati

I diversi tipi di dataset utilizzano meccanismi differenti per il collegamento alle informazioni del database sottostante. Questi meccanismi differenti, a loro volta, sono quelli che comportano le principali differenze nell'architettura delle applicazioni database che è possibile costruire. Esistono quattro meccanismi di base per il collegamento ai dati:

- Connessione diretta a un server di database. La maggior parte dei dataset utilizza un discendente di *TCustomConnection* per rappresentare la connessione a un server di database.
- Utilizzo di un file dedicato su disco. I dataset client supportano la capacità di operare con un file dedicato su disco. Quando si opera con un file dedicato non è

necessario alcun componente connection separato perché lo stesso dataset client sa come leggere dal file e scrivere in esso.

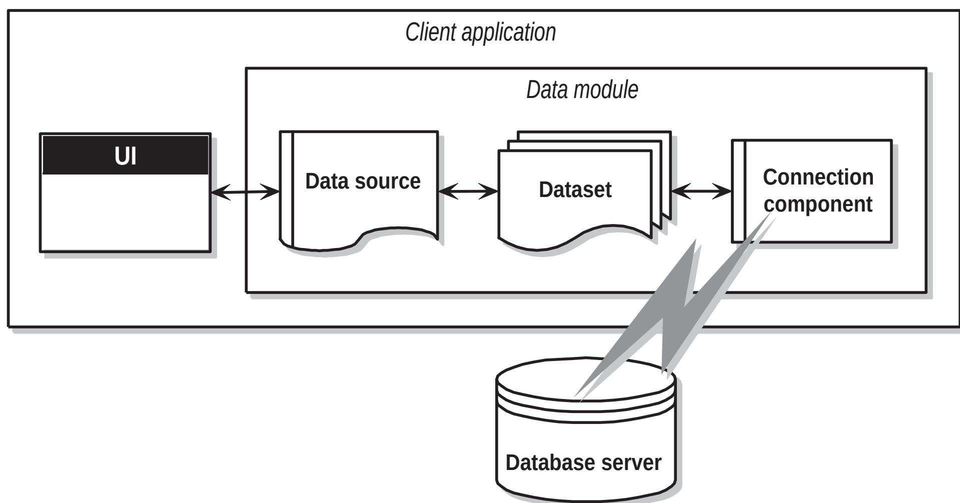
- Collegamento a un altro dataset. I dataset client possono operare con dati forniti da un altro dataset. Un componente *TDataSetProvider* agisce da intermediario tra il dataset client e il proprio dataset sorgente. Questo provider di dataset può risiedere nello stesso modulo dati del dataset client o può fare parte di un application server in esecuzione su un'altra macchina. Se il provider fa parte di un application server, è necessario anche un discendente speciale di *TCustomConnection* per rappresentare la connessione all'application server.
- Ottenimento dei dati da un oggetto RDS DataSpace. I dataset ADO possono utilizzare un componente *TRDSConnection* per smistare i dati nelle applicazioni database multi-tier costruite utilizzando application server basati su ADO.

Talvolta, questi meccanismi possono essere combinati in una singola applicazione.

Connessione diretta a un server di database

L'architettura di database più comune è quella in cui il dataset utilizza un componente connection per stabilire una connessione con un server di database. Il dataset quindi preleva i dati direttamente dal server e invia le modifiche direttamente al server. L'architettura è illustrata nella Figura 18.2.

Figura 18.2 Connessione diretta a un server di database



Ogni tipo di dataset utilizza il proprio tipo di componente connection, che rappresenta un singolo meccanismo di accesso ai dati:

- Se il dataset è un dataset BDE come TTable, *TQuery* o *TStoredProc*, il componente connection è un oggetto *TDataBase*. Si collega il dataset al componente database impostandone la proprietà *Database*. Non è necessario aggiungere esplicitamente un componente databse quando si usa un dataset BDE. Se si imposta la proprietà

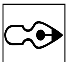
DatabaseName del dataset, in fase di esecuzione viene creato automaticamente un componente database.

- Se il dataset è un dataset ADO come *TADODataSet*, *TADOTable*, *TADOQuery* o *TADOStoredProc*, il componente connection è un oggetto *TADOConnection*. Si collega il dataset al componente connection ADO impostandone la proprietà *ADOConnection*. Come con i dataset BDE, non è necessario aggiungere esplicitamente il componente connection: è possibile invece impostare la proprietà *ConnectionString* del dataset.
- Se il dataset è un dataset *dbExpress* come *TSQLDataSet*, *TSQLTable*, *TSQLQuery* o *TSQLStoredProc*, il componente connection è un oggetto *TSQLConnection*. Si collega il dataset al componente connection SQL impostandone la proprietà *SQLConnection*. Quando si utilizzano dataset *dbExpress*, è necessario aggiungere esplicitamente il componente connection. Un'altra differenza fra il dataset *dbExpress* e gli altri dataset è che i dataset *dbExpress* sono sempre a sola lettura e unidirezionali: ciò significa che è possibile spostarsi fra i record solo in base alla loro sequenza e non è possibile utilizzare i metodi del dataset che supportano le modifiche.
- Se il dataset è un dataset InterBase Express come *TIBDataSet*, *TIBTable*, *TIBQuery* o *TIBStoredProc*, il componente connection è un oggetto *TIBDatabase*. Si collega il dataset al componente di database IB impostandone la proprietà *Database*. Come con i dataset *dbExpress*, è necessario aggiungere esplicitamente il componente connection.

Oltre ai componenti elencati sopra, è possibile utilizzare un dataset client specializzato come *TBDEClientDataSet*, *TSQLClientDataSet* o *TIBClientDataSet* con un componente database connection. Se si utilizza uno di questi dataset client, specificare il tipo appropriato di componente connection come valore della proprietà *DBConnection*.

Benché ogni tipo di dataset utilizzi un componente connection diverso, tutti quanti svolgono la maggior parte delle stesse attività e rendono disponibili molte proprietà, metodi e eventi che tutti hanno in comune. Per ulteriori informazioni sulle similitudini tra i diversi componenti di connessione ai database, vedere il Capitolo 21, "Connessione ai database".

Questa architettura rappresenta sia un'applicazione single-tier sia un'applicazione two-tier, a seconda che il server di database sia un database locale o un server di database remoto. La logica che tratta le informazioni di database è nella stessa applicazione che implementa l'interfaccia utente, sebbene isolata in un modulo dati.

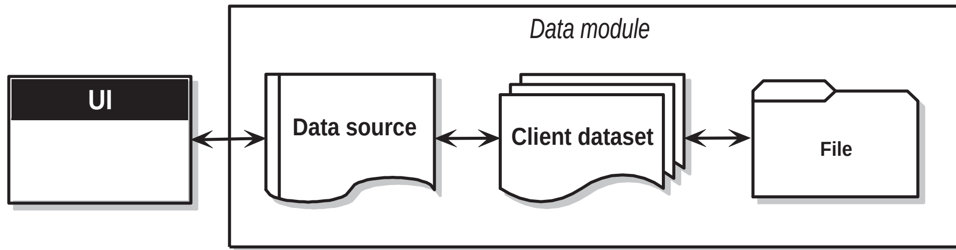


I componenti connection o i driver necessari per creare applicazioni two-tier non sono disponibili in tutte le versione di C++Builder.

Utilizzo di un file dedicato su disco

La forma più semplice di un'applicazione database che è possibile scrivere non utilizza affatto un server di database. Invece, con MyBase, utilizza la capacità dei dataset client di salvare il proprio contenuto in un file e di caricare i dati da un file. Questa architettura è illustrato nella Figura 18.3:

Figura 18.3 Un'applicazione database basata su file



Se si utilizza questo approccio basato su file, l'applicazione scrive le modifiche sul disco utilizzando il metodo *SaveToFile* del dataset client. *SaveToFile* richiede un parametro, il nome del file da creare (o da sovrascrivere) che contiene la tabella. Quando si vuole leggere una tabella precedentemente scritta utilizzando il metodo *SaveToFile*, utilizzare il metodo *LoadFromFile*. Anche *LoadFromFile* richiede un parametro, il nome del file che contiene la tabella.

Se si caricano i dati e li si salva utilizzando sempre lo stesso file, è possibile utilizzare la proprietà *FileName* invece dei metodi *SaveToFile* e *LoadFromFile*. Quando *FileName* è impostata a un nome di file valido, i dati vengono automaticamente caricati dal file quando si apre il dataset client e salvati sul file quando si chiude il dataset client.

Questa semplice architettura basata su file è un'applicazione single-tier. La logica che tratta le informazioni di database è nella stessa applicazione che implementa l'interfaccia utente, sebbene isolata in un modulo dati.

L'approccio basato su file ha il vantaggio della semplicità. Non occorre installare, configurare o distribuire alcun server di database. (anche se il dataset client necessita di midas.dll). Non c'è alcuna necessità di licenze di postazione o di gestione del database.

Inoltre, alcune versioni di C++Builder permettono di convertire documenti arbitrari XML in pacchetti dati utilizzati da un dataset client. Pertanto, l'approccio basato su file può essere utilizzato per operare con documenti XML oltre che con dataset dedicati. Per informazioni sulla conversione tra documenti XML e pacchetti dati di dataset client, vedere il Capitolo 30, "Uso di XML nelle applicazioni database".

L'approccio basato su file non offre alcun supporto per più utenti. Il dataset deve essere dedicato completamente all'applicazione. I dati vengono salvati in un file su disco e caricati in un momento successivo, ma non c'è alcuna protezione nativa atta ad impedire a più utenti di sovrascrivere l'un l'altro gli archivi dati.

Per ulteriori informazioni sull'utilizzo di un dataset client con dati memorizzati su disco, vedere "Uso di un dataset client con dati basati su file" a pagina 27-35.

Collegamento a un altro dataset

Esistono dataset client specializzati che utilizzano BDE o *dbExpress* per collegarsi a un server di database. Questi dataset client specializzati sono, infatti, componenti compositi che dispongono di un altro dataset interno per accedere ai dati e di un

componente provider interno per assemblare i dati dal dataset sorgente e riapplicare gli aggiornamenti al server di database. Questi componenti compositi risultano un po' più pesanti in termini di gestione, ma offrono alcuni vantaggi:

- I dataset client rappresentano il modo più affidabile per operare con gli aggiornamenti in cache. Per impostazione predefinita, altri tipi di dataset inviano le modifiche direttamente al server di database. Utilizzando un dataset che memorizza gli aggiornamenti nella cache locale e li applica tutti in un secondo momento con una singola transazione è possibile ridurre il traffico di rete. Per informazioni sui vantaggi derivanti dall'utilizzo di dataset client per memorizzare gli aggiornamenti in cache, vedere "Utilizzo di un dataset client per registrare in cache gli aggiornamenti" a pagina 27-17.
- I dataset client possono applicare le modifiche direttamente a un server di database se il dataset è a sola lettura. Se si utilizza *dbExpress*, questo è l'unico modo per modificare i dati nel dataset (ed è anche l'unico modo per spostarsi liberamente tra i dati). Anche se non si utilizza *dbExpress*, i risultati di alcune query e di tutte le procedure registrate sono a sola lettura. L'utilizzo di un dataset client fornisce un metodo standard per rendere modificabili tali dati.
- Poiché i dataset client possono operare direttamente con file dedicati su disco, l'utilizzo di un dataset client può essere combinato con un modello basato su file in modo da ottenere un'applicazione "briefcase " flessibile. Per informazioni sul modello briefcase, vedere "Combinazione dei vari approcci" a pagina 18-15.

Oltre a questi dataset client specializzati, esiste un dataset client generico (*TClientDataSet*), che non dispone di un dataset interno e di un provider di dataset. Benché *TClientDataSet* non abbia alcun meccanismo interno per l'accesso ai database, è possibile connetterlo a un altro dataset esterno da cui preleva i dati e a cui invia gli aggiornamenti. Benché questo approccio sia un po' più complicato, a volte potrebbe essere quello migliore:

- Poiché il dataset sorgente e il provider di dataset sono esterni, si ha un maggior controllo sul modo in cui prelevano i dati e applicano gli aggiornamenti. Ad esempio, il componente provider rende disponibili molti eventi che non sono disponibili se si utilizza un dataset client specializzato per accedere ai dati.
- Se il dataset sorgente è esterno, è possibile collegarlo mediante una relazione master/detail con un altro dataset. Un provider esterno converte automaticamente questa configurazione in un singolo dataset con dettagli annidati.Se il dataset sorgente è interno, è impossibile creare set di dettaglio annidati secondo questa modalità.
- La connessione di un dataset client a un dataset esterno è un'architettura che può essere estesa progressivamente fino a più tier. Poiché il processo di sviluppo può diventare sempre più complesso e costoso man mano che aumenta il numero di tier, potrebbe essere opportuno iniziare a sviluppare l'applicazione come single-tier o two-tier. Via via che la quantità di dati, il numero di utenti e il numero di applicazioni differenti che accedono ai dati cresce, si può passare gradualmente a un'architettura multi-tiered. Se si presume che si arriverà a utilizzare un'architettura multi-tier, può riconvenienete iniziare lo sviluppo utilizzando un dataset client e un dataset sorgente esterno. In questo modo, quando si sposterà l'accesso ai dati e la logica di gestione su un tier intermedio, l'investimento fatto in

termini di sviluppo risuletà protetto poiché il codice potrà essere riutilizzato man mano che l'applicazione cresce.

- *TClientDataSet* può collegarsi a qualsiasi dataset sorgente. Ciò significa che è possibile utilizzare dataset custom (componenti forniti da terze parti) per i quali non esiste alcun dataset client specializzato corrispondente. Alcune versioni di C++Builder includono anche componenti provider particolari che connettono un dataset client a un documento XML invece che a un altro dataset. Il funzionamento è analogo alla connessione di un dataset client a un altro dataset (sorgente), salvo che il provider XML utilizza un documento XML al posto di un dataset. Per informazioni su questi provider XML, vedere "Impiego di un documento XML come origine di un provider" a pagina 30-8.)

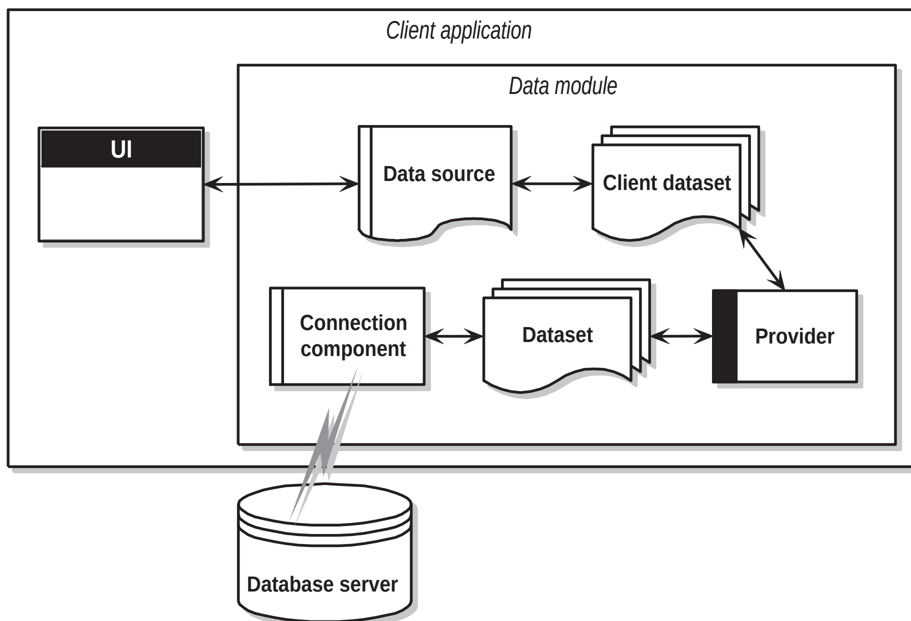
Esistono due versioni dell'architettura che connette un dataset client a un dataset esterno:

- Connessione di un dataset client a un altro dataset nella stessa applicazione.
- Utilizzo di un'architettura multi-tier.

Connessione di un dataset client a un altro dataset nella stessa applicazione

Utilizzando un componente provider, è possibile connettere *TClientDataSet* a un altro dataset (sorgente). Il provider assembla le informazioni del database in pacchetti dati trasportabili (che possono essere utilizzati dai dataset client) e applica gli aggiornamenti ricevuti in pacchetti delta (creati dai dataset client) a un server di database. Questa architettura è illustrata nella Figura 18.4.

Figura 18.4 Architettura che combina un dataset client e un altro dataset



Questa architettura rappresenta sia un'applicazione single-tier sia un'applicazione two-tier, a seconda che il server di database sia un database locale o un server di database remoto. La logica che tratta le informazioni di database è nella stessa applicazione che implementa l'interfaccia utente, sebbene isolata in un modulo dati.

Per collegare il dataset client con il provider, impostarne la proprietà *ProviderName* al nome del componente provider. Il provider deve essere nello stesso modulo dati del dataset client. Per collegare il provider con il dataset sorgente, impostarne la proprietà *DataSet*.

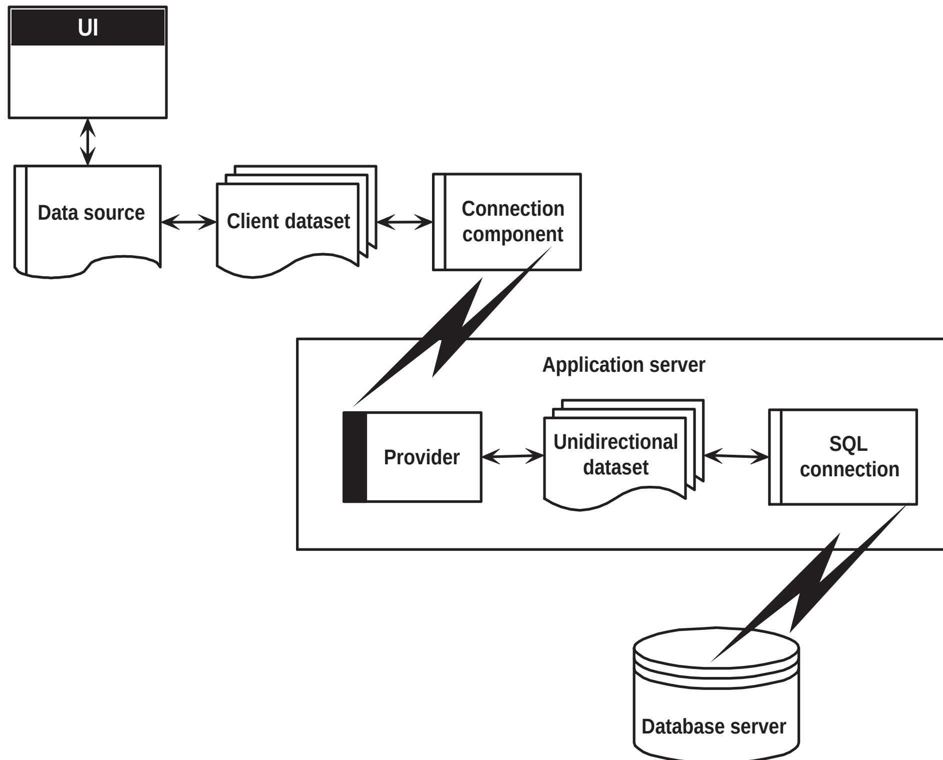
Una volta che il dataset client è collegato con il provider e il provider è collegato con il dataset sorgente, questi componenti gestiscono automaticamente tutti i dettagli necessari per prelevare, visualizzare e consentire lo spostamento attraverso i record del database (partendo dall'assunto che dataset sorgente sia connesso a un database). Per applicare al database le modifiche apportate dall'utente, è necessario solo chiamare il metodo *ApplyUpdates* del dataset client.

Per ulteriori informazioni sull'utilizzo di un dataset client con un provider, vedere "Uso di un dataset client con un provider di dati" a pagina 27-26.

Utilizzo di un'architettura multi-tier

Quando le informazioni del database comprendono relazioni complesse tra più tabelle, o quando il numero di client aumenta, potrebbe essere necessario usare un'applicazione multi-tier. Le applicazioni multi-tier hanno tier intermedi fra l'applicazione client e il server di database. Questa architettura è illustrata nella Figura 18.5.

Figura 18.5 Architettura multi-tier di un database



La figura precedente rappresenta un'applicazione three-tier.La logica che tratta le informazioni di database è su un sistema, o tier, separato. Questo tier intermedio centralizza la logica che governa le interazioni col database centralizzando in tal modo il controllo sulle relazioni tra i dati. Ciò permette a varie applicazioni client di utilizzare gli stessi dati, assicurando al contempo la congruenza della logica dei dati. Permette anche di ottenere applicazioni client più piccole perché gran parte dell'elaborazione è demandata al tier intermedio. Queste applicazioni client più piccole risultano più facili da installare, configurare e gestire. Le applicazioni multi-tier possono anche migliorare le prestazioni distribuendo le attività di elaborazione dei dati su parecchi sistemi.

L'architettura multi-tier è molto simile al modello precedente. Si differenzia principalmente per il fatto che il dataset sorgente, che si collega al server di database, e il provider, che agisce come intermediario tra quel dataset sorgente e il dataset client, sono stati spostati entrambi in un'applicazione separata. Questa e applicazione separata è detta application server (o talvolta "data broker remoto").

Poiché il provider è stato spostato in un'applicazione separata, il dataset client non può più essere collegato al dataset sorgente impostando semplicemente la sua proprietà *ProviderName*. Inoltre, deve utilizzare alcuni tipi di componente connection per individuare e connettersi all'applicazione server.

Esistono molti tipi di componenti connection in grado di connettere un dataset client a un application server. Sono tutti discendenti di *TCustomRemoteServer* e si differenziano principalmente per il protocollo di comunicazione utilizzato (TCP/IP, HTTP, DCOM, SOAP o CORBA). Collegare il dataset client con il suo componente connection impostandone la proprietà *RemoteServer*.

Il componente connection stabilisce una connessione con l'application server e restituisce un'interfaccia che viene utilizzata dal dataset client per chiamare il provider specificato mediante la sua proprietà *ProviderName*. Ogni volta che il dataset client chiama l'application server, passa il valore di *ProviderName* e il server applicazione inoltra la chiamata al provider.

Per ulteriori informazioni sulla connessione di un dataset client a un application server, consultare il Capitolo 29, "Creazione di applicazioni multi-tier".

Combinazione dei vari approcci

Le sezioni precedenti descrivono numerose architetture che è possibile utilizzare quando si scrivono applicazioni database. Non c'è alcun motivo, tuttavia, che impedisce di combinare due o più delle architetture disponibili in una singola applicazione. Infatti, alcune combinazioni possono essere estremamente potenti.

Ad esempio, è possibile combinare l'architettura basata su disco descritta nel paragrafo "Utilizzo di un file dedicato su disco" a pagina 18-9 con un altro approccio come quelli descritti in "Connessione di un dataset client a un altro dataset nella stessa applicazione" a pagina 18-12 oppure in "Utilizzo di un'architettura multi-tier" a pagina 18-13. Queste combinazioni sono semplici perché tutti i modelli utilizzano un dataset client per rappresentare i dati che vengono visualizzati nell'interfaccia utente. Il risultato è definito modello briefcase (o talvolta modello non connesso, o mobile computing).

Il modello briefcase è utile in una situazioni come quella descritta di seguito: Un database aziendale contiene i dati relativi ai contatti dei clienti che i rappresentanti possono utilizzare e aggiornare mentre operano sul campo. Durante la loro permanenza in sede, i rappresentanti scaricheranno le informazioni dal database. Successivamente, lavoreranno sui dati con i loro computer portatili durante gli spostamenti attraverso il paese e potranno anche aggiornare i record mentre saranno presso la sede dei nuovi o dei clienti esistenti. Una volta tornati in sede, i rappresentanti dovranno caricare nel database aziendale le proprie modifiche ai dati, affinché siano utilizzabili da tutti.

Durante le operazioni in sede, il dataset client di un'applicazione che segue il modello briefcase reperirà i dati utilizzando un provider. Il dataset client è quindi connesso al server di database e può, tramite il provider, prendere i dati del server e restituire gli aggiornamenti al server. Prima di scollegarsi dal provider, il dataset client salva l'istantanea delle informazioni in un file su disco. Durante le operazioni fuori sede, il dataset client carica i dati dal file e salva le eventuali modifiche in quel file. Infine, tornato di nuovo in sede, il dataset client si riconnette al provider in modo da applicare gli aggiornamenti al server di database o aggiornare la propria istantanea dei dati.

Progettazione dell’interfaccia utente

La pagina Data Controls della Component palette fornisce una serie di controlli associati ai dati che rappresentano i dati dei campi in un record del database, e può permettere agli utenti di modificare questi dati e di inoltrare le modifiche al database stesso. Usando i controlli associati ai dati, è possibile costruire l’interfaccia utente (UI) dell’applicazione database in modo che le informazioni siano visibili e accessibili agli utenti. Per ulteriori informazioni sui controlli data-aware, consultare il Capitolo 19, “Uso dei controlli dati”.

Oltre ai controlli dati di base, potrebbe essere necessario introdurre nell’interfaccia utente altri elementi:

- Potrebbe essere necessario che l’applicazione analizzi i dati contenuti in un database. Le applicazioni che analizzano dati non si limitano semplicemente a visualizzare i dati in un database, ma compongono anche le informazioni in formati atti ad aiutare gli utenti a percepire il significato intrinseco di quei dati.
- Potrebbe essere necessario stampare dei report in modo da fornire una copia cartacea delle informazioni visualizzate nell’interfaccia utente.
- Potrebbe essere necessario creare un’interfaccia utente visualizzabile tramite un Web browser. Alcune semplici applicazioni database basate su Web sono descritte in “Utilizzo nelle risposte di informazioni di database” a pagina 33-18. Inoltre, è possibile combinare l’approccio basato su Web con l’architettura multi-tier, come descritto in “Scrittura di applicazioni client basate su Web”.

Analisi dei dati

Alcune applicazioni database non presentano le informazioni del database direttamente all’utente. Esse analizzano e riassumono, invece, le informazioni dai database in modo che gli utenti possano trarre conclusioni dai dati.

Il componente *TDBChart* nella pagina Data Controls della Component palette permette di presentare le informazioni del database in un formato grafico che consente agli utenti di cogliere rapidamente il senso delle informazioni del database.

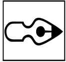
Inoltre, alcune versioni di C++Builder includono nella Component palette una pagina Decision Cube. Questa pagina contiene sei componenti che permettono di eseguire l’analisi e il confronto incrociato dei dati durante la creazione di applicazioni per il supporto decisionale. Per ulteriori informazioni sull’uso dei componenti Decision Cube, consultare il Capitolo 20, “Uso dei componenti di supporto decisionale”.

Se si desidera costruire propri componenti in grado di visualizzare riepiloghi di dati in base a diversi criteri di raggruppamento, è possibile usare le aggregazioni di manutenzione con un dataset client. Per ulteriori informazioni sull’uso delle aggregazioni di manutenzione, consultare “Uso delle aggregazioni di manutenzione” a pagina 27-12.

Scrittura di report

Per consentire agli utenti di stampare le informazioni del database dai dataset nell'applicazione, si possono usare i componenti report della pagina QReport della Component palette. Usando questi componenti, si possono costruire in modo visuale report suddivisi in bande per presentare e sintetizzare le informazioni contenute nelle tabelle del database. È possibile aggiungere riepiloghi per raggruppare intestazioni o piè pagina in modo da analizzare i dati in base a criteri di raggruppamento.

Per creare un report per l'applicazione, selezionare l'icona QuickReport dalla finestra di dialogo New Items. Selezionare File|New\Other dal menu principale e richiamare la pagina Business. Per avviare il wizard, fare doppio clic sull'icona QuickReport Wizard.



Per un esempio di come utilizzare i componenti presenti sulla pagina QReport, esaminare il programma di esempio QuickReport fornito con C++Builder.

Capitolo

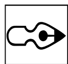
19

Uso dei controlli dati

La pagina Data Controls della Component palette fornisce un insieme di controlli data-aware per rappresentare i dati dei campi di un record di un database e, se il dataset lo permette, consente agli utenti di modificare quei dati e di registrare le modifiche nel database. Collocando sulle schede dell'applicazione database questi controlli per dati è possibile costruire l'interfaccia utente (UI) dell'applicazione database in modo da rendere visibili e accessibili agli utenti tali informazioni.

I controlli data-aware che vengono aggiunti all'interfaccia utente dipendono da numerosi fattori, tra cui:

- Il tipo di dati da visualizzare. È possibile scegliere fra controlli progettati per visualizzare e modificare un testo non formattato, controlli che operano con testo formattato, controlli per la grafica, elementi multimediali e così via. I controlli che visualizzano vari tipi di informazioni sono descritti in "Visualizzazione di un singolo record" a pagina 19-8.
- Come si vogliono organizzare le informazioni. È possibile scegliere di visualizzare le informazioni contenute in un singolo record o di elencare le informazioni contenute in più record utilizzando una griglia. Il paragrafo "Scelta del modo in cui organizzare i dati" a pagina 19-8 descrive alcune possibilità.
- Il tipo di dataset che fornisce i dati ai controlli. Si potrebbero utilizzare controlli che riflettano le limitazioni del dataset sottostante. Ad esempio, non si utilizzerà mai una griglia con un dataset unidirezionale perché i dataset unidirezionali possono fornire solo un singolo record per volta.
- Come (o se) si desidera consentire agli utenti di spostarsi tra i record dei dataset e aggiungere o modificare i dati. Si potrebbero aggiungere controlli o meccanismi personali per spostarsi tra i record e modificarli oppure si potrebbe utilizzare un controllo nativo come programma per la navigazione tra i dati. Per ulteriori informazioni sull'utilizzo di un componente per la navigazione tra i dati, vedere "Spostamento e gestione dei record" a pagina 19-31.



Controlli data-aware più complessi per il supporto decisionale sono descritti nel Capitolo 20, "Uso dei componenti di supporto decisionale".

Indipendentemente dai controlli data-aware che si sceglie di aggiungere all'interfaccia, si applicano alcune funzionalità comuni. Queste sono descritti nel prosieguo del capitolo.

Uso delle funzioni comuni ai controlli dati

Le seguenti operazioni sono comuni alla maggior parte dei controlli dati:

- Associazione di un controllo dati a un dataset
- Modifica e aggiornamento dei dati
- Disabilitazione e abilitazione della visualizzazione dei dati
- Aggiornamento della visualizzazione dei dati
- Abilitazione degli eventi mouse, tastiera e timer

I controlli dati consentono di visualizzare e di modificare i campi dati associati al record attivo in un dataset. La Tabella 19.1 riepiloga i controlli dati che appaiono nella pagina Data Controls della Component palette.

Tabella 19.1 Controllo dati

Controllo dati	Descrizione
TDBGrid	Mostra informazioni da una sorgente dati in formato tabellare. Le colonne della griglia corrispondono alle colonne della tabella associata o al dataset della query. Le righe della griglia corrispondono ai record.
TDBNavigator	Consente di spostarsi tra i record di dati di un dataset, di aggiornare i record, di confermarli, di cancellare le modifiche e di aggiornare la visualizzazione dei dati.
TDBText	Mostra dati tratti da un campo sotto forma di etichetta.
TDBEdit	Mostra dati tratti da un campo in una casella di testo.
TDBMemo	Mostra dati tratti da un campo memo o BLOB in una casella di testo multirighe.
TDBImage	Mostra grafici di un campo dati in una casella grafica.
TDBListBox	Mostra una lista di elementi da cui effettuare una scelta per aggiornare un campo nel record dati attivo.
TDBComboBox	Mostra una lista di elementi fra cui scegliere per aggiornare un campo e consente inoltre di immettere direttamente un testo come in una casella di testo associata ai dati standard.
TDBCheckBox	Mostra una casella di controllo che indica il valore di un campo booleano.
TDBRadioGroup	Mostra una serie di opzioni per il campo, che si escludono reciprocamente.
TDBLookupListBox	Mostra una lista di elementi cercati in un altro dataset in base al valore di un campo.
TDBLookupComboBox	Mostra una lista di elementi cercati in un altro dataset in base al valore di un campo; consente, inoltre, di immettere direttamente un testo come in una casella standard di testo associata ai dati.

Tabella 19.1 Controllo dati (continua)

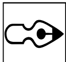
Controllo dati	Descrizione
TDBCtrlGrid	Mostra, all'interno di una griglia, un insieme di controlli associati ai dati configurabili e ripetitivi.
TDBRichEdit	Mostra dati formattati di un campo in una casella di testo.

I controlli dati vengono associati ai dati in fase di progetto. Quando si associa un controllo dati con un dataset attivo durante la creazione di un'applicazione, nel controllo è possibile vedere immediatamente dati dinamici. In fase di progetto è possibile utilizzare il Fields Editor per effettuare lo scorrimento attraverso un dataset e verificare che l'applicazione visualizzi i dati correttamente, senza doverla compilare ed eseguire. Per ulteriori informazioni su Fields Editor, vedere "Creazione di campi persistenti" a pagina 23-4.

In fase di esecuzione i controlli dati visualizzano i dati che l'utente può modificare tramite il controllo, nel caso l'applicazione, il controllo e il dataset lo consentano.

Associazione di un controllo dati a un dataset

I controlli dati si collegano ai dataset usando un datasource. Un componente datasource (*TDataSource*) si comporta come un elemento di connessione tra il controllo e un dataset contenente dati. Ogni controllo data-aware deve essere associato a un componente datasource per poter avere dei dati da visualizzare e trattare. Analogamente, tutti i dataset devono essere associati a un componente datasource affinché i relativi dati siano visualizzati e trattati nei controlli data-aware di una scheda.



I componenti Datasource sono necessari anche per collegare dataset non annidati nelle relazioni master-detail.

Per associare un controllo dati a un dataset:

1. Collocare un dataset in un modulo dati (o su una scheda) impostarne in modo adeguato le proprietà.
2. Collocare un datasource nello stesso modulo dati (o scheda). Utilizzando l'Object Inspector, impostarne la proprietà *DataSet* al nome del dataset collocato al punto 1.
3. Inserire un controllo dati dalla pagina Data Access della Component palette in una scheda.
4. Utilizzando l'Object Inspector, impostare la proprietà *DataSource* del controllo al datasource del componente collocato al punto 2.
5. Impostare la proprietà *DataField* del controllo al nome di un campo da visualizzare, o selezionare un nome di campo dall'elenco a comparsa della proprietà. Questa fase non vale per a *TDBGrid*, *TDBCtrlGrid*, e *TDBNavigator* perché accedono a tutti i campi disponibili nel dataset.
6. Impostare la proprietà *Active* del dataset a **true** per visualizzare i dati del controllo.

Modifica del dataset associato in esecuzione

Nell'esempio precedente, il datasource è stato associato al proprio dataset impostando la proprietà *DataSet* in fase di progettazione. In fase di esecuzione, se occorre, è possibile scambiare il dataset di un componente datasource. Ad esempio, il codice seguente commuta il dataset del componente datasource *CustSource* tra i componenti dataset di nome *Customers* e *Orders*:

```
if (CustSource->DataSet == Customers)
  CustSource->DataSet = Orders;
else
  CustSource->DataSet = Customers;
```

È possibile anche impostare la proprietà *DataSet* al dataset di un'altra scheda in modo da sincronizzare i controlli dati sulle due schede. Per esempio:

```
void __fastcall TForm2::FormCreate(TObject *Sender)
{
  DataSource1->DataSet = Form1->Table1;
}
```

Attivazione e disattivazione del datasource

Il datasource ha una proprietà *Enabled* che determina se è connesso al proprio dataset. Se *Enabled* è **true**, il datasource è connesso al dataset.

È possibile sconnettere temporaneamente un singolo datasource dal suo dataset impostando *Enabled* su **false**. Quando *Enabled* è **false**, tutti i controlli dati connessi al componente datasource diventano vuoti e inattivi finché *Enabled* non è impostato a **true**. È consigliabile, tuttavia, controllare l'accesso a un dataset tramite i metodi *DisableControls* e *EnableControls* di un componente dataset in quanto essi influenzano tutti i datasource collegati.

Risposta alle modifiche mediate dal datasource

Poiché il datasource fornisce il collegamento tra il controllo dati e il proprio dataset, esso funge da intermediario per tutte le comunicazioni che avvengo tra i due. Di solito, il controllo data-aware risponde automaticamente alle modifiche nel dataset. Tuttavia, se l'interfaccia utente utilizza controlli che non sono data-aware, è possibile utilizzare gli eventi di un componente datasource per fornire manualmente lo stesso tipo di risposta.

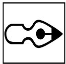
L'evento *OnDataChange* si verifica ogni volta che i dati in un record possono aver subito un cambiamento, sia per la modifica del campo sia perché il cursore passa su un nuovo record. Questo evento risulta utile per assicurarsi che il controllo rifletta i valori correnti del campo nel dataset, in quanto viene attivato da tutte le modifiche. Di solito, un gestore di evento *OnDataChange* viene utilizzato per aggiornare il valore di un controllo non data-aware che visualizza i dati di un campo.

L'evento *OnUpdateData* si verifica nel momento in cui si stanno per registrare i dati nel record corrente. Per esempio, si verifica un evento *OnUpdateData* dopo che viene chiamato Post, ma prima che i dati vengano effettivamente registrati nel server di database sottostante o nella cache locale.

L'evento *OnStateChange* si verifica quando cambia lo stato del dataset. Quando si verifica questo evento, è possibile esaminare la proprietà *State* del dataset per determinarne lo stato corrente.

Per esempio, il seguente gestore di evento *OnStateChange* attiva o disattiva i pulsanti o le voci di menu in base allo stato corrente:

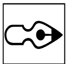
```
void __fastcall TForm1::DataSource1StateChange(TObject *Sender)
{
  CustTableActivateBtn->Enabled = (CustTable->State == dsInactive);
  CustTableEditBtn->Enabled = (CustTable->State == dsBrowse);
  CustTableCancelBtn->Enabled = (CustTable->State == dsInsert ||
                                 CustTable->State == dsEdit ||
                                 CustTable->State == dsSetKey);
  ⋮
}
```



Per ulteriori informazioni sugli stati del dataset, consultare "Determinazione degli stati del dataset" a pagina 22-3.

Modifica e aggiornamento dei dati

Tutti i controlli dati, tranne il navigatore, visualizzano i dati di un campo database. Inoltre, possono essere utilizzati per modificare e aggiornare i dati finché il dataset sottostante lo permette.



I dataset unidirezionali non consentono mai all'utente di modificare e aggiornare i dati.

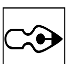
Abilitazione dell'editing nei controlli all'atto dell'immissione dati

Un dataset deve essere in stato *dsEdit* perché sia possibile modificarne i dati. Se la proprietà *AutoEdit* del datasource è **true** (impostazione predefinita), il controllo dati gestisce l'attività di porre il dataset in modalità *dsEdit* non appena l'utente prova a modificarne i dati.

Se *AutoEdit* è **false**, è necessario fornire un meccanismo alternativo per porre il dataset in modalità editing. Un possibile meccanismo consiste nell'utilizzare un controllo *TDBNavigator* con un pulsante *Edit* che consenta esplicitamente all'utente di porre il dataset in modalità editing. Per ulteriori informazioni su *TDBNavigator*, vedere "Spostamento e gestione dei record" a pagina 19-31. In alternativa, è possibile scrivere del codice che chiami il metodo *Edit* del dataset quando si desidera porre il dataset in modalità editing.

Modifica dei dati in un controllol

Un controllo dati può registrare le modifiche apportate al dataset associato solo se la proprietà *CanModify* del dataset è **true**. *CanModify* è sempre **false** per i dataset unidirezionali. Alcuni dataset hanno una proprietà *ReadOnly* che consente di specificare se *CanModify* è **true**.



Il fatto che un dataset client possa aggiornare i dati dipende dal fatto che la tabella del database sottostante consenta o meno gli aggiornamenti.

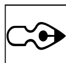
Anche se la proprietà *CanModify* del dataset è impostata a **true**, affinché il controllo possa registrare gli aggiornamenti nella tabella di database, è necessario impostare a **true** anche la proprietà *Enabled* del datasource che connette il dataset al controllo. La proprietà *Enabled* del datasource determina se il controllo può visualizzare i valori dei campi del dataset e, di conseguenza, se un utente può modificare i valori e registrare le modifiche. Se *Enabled* è **true** (impostazione predefinita), i controlli possono visualizzare i valori dei campi.

Infine, è possibile controllare anche se l'utente può apportare modifiche ai dati visualizzati nel controllo. La proprietà *ReadOnly* del controllo dati determina se un utente può modificare i dati visualizzati dal controllo. Se è **false** (impostazione predefinita), gli utenti possono modificare i dati. Ovviamente, sarà bene assicurarsi che la proprietà *ReadOnly* del controllo sia **true** quando la proprietà *CanModify* del dataset è **false**. In caso contrario, si darà falsamente all'utente l'impressione che possa influire sui dati nella tabella del database sottostante.

In tutti i controlli dati, ad eccezione di *TDBGrid*, le modifiche apportate a un campo vengono copiate nel componente field sottostante in un dataset quando viene premuto il tasto *Tab* dal controllo. Se in un campo si preme *Esc* prima di *Tab*, il controllo dati abbandona le modifiche e ripristina il valore del campo antecedente alle modifiche.

In *TDBGrid*, le modifiche vengono confermate solo quando si passa a un altro record; per annullare tutte le modifiche apportate a un qualsiasi record di un campo, è possibile premere *Esc* prima di passare a un altro record.

Quando un record viene registrato, C++Builder controlla tutti i componenti data-aware associati al dataset, per verificare se il loro stato è cambiato. Se sorge un problema di aggiornamento di un campo contenente dati modificati, C++Builder solleva un'eccezione e il record non viene modificato.



Se l'applicazione memorizza gli aggiornamenti nella cache (ad esempio, utilizzando un dataset client), tutte le modifiche vengono inviate a una cache interna. Queste modifiche non vengono applicate alla tabella di database sottostante finché non si chiama il metodo *ApplyUpdates* del dataset.

Disabilitazione e abilitazione della visualizzazione dei dati

Quando l'applicazione elabora in sequenza un dataset o esegue una ricerca, è opportuno impedire temporaneamente l'aggiornamento dei valori visualizzati nei controlli associati ai dati a ogni variazione del record attivo. Impedendo l'aggiornamento dei valori, l'operazione sequenziale o la ricerca risultano più veloci e si evita il fastidioso sfarfallio dello schermo.

DisableControls è un metodo dei dataset che disabilita la visualizzazione di tutti i controlli associati ai dati legati a un dataset. Non appena sono terminate l'elaborazione sequenziale o la ricerca, l'applicazione deve chiamare immediatamente il metodo *EnableControls* del dataset per abilitare nuovamente la visualizzazione nei controlli.

Di solito, i controlli vengono disabilitati prima di iniziare un processo iterativo. Quest'ultimo deve aver luogo all'interno di un'istruzione **try...finally**, in modo che

sia possibile riabilitare i controlli anche se si verifica un'eccezione durante l'elaborazione. La clausola **finally** chiamerà *EnableControls*. Il codice seguente illustra come utilizzare *DisableControls* e *EnableControls* in questo modo:

```
CustTable->DisableControls();
try
{
  // cycle through all records of the dataset
  for (CustTable->First(); !CustTable->EOF; CustTable->Next())
  {
  // Process each record here
  ⋮
  }
}
__finally
{
  CustTable->EnableControls();
}
```

Aggiornamento della visualizzazione dei dati

Il metodo *Refresh* di un dataset svuota i buffer locali e recupera nuovamente i dati per un dataset aperto. Questo metodo può essere usato per aggiornare la visualizzazione nei controlli data-aware, qualora si ritenga che i dati sottostanti possano essere stati modificati da altre applicazioni che hanno accesso simultaneo agli stessi dati. Se si stanno utizzando gli aggiornamenti tramite cache, prima di aggiornare il dataset è necessario applicare tutti gli aggiornamenti che il dataset ha attualmente memorizzato nella cache.

L'aggiornamento può talvolta comportare risultati imprevisti. Per esempio, se un utente visualizza un record che è stato cancellato da un'altra applicazione, il record scomparirà nel momento in cui l'applicazione chiama Refresh. I dati possono, inoltre, sembrare diversi se un altro utente cambia un record nel periodo di tempo che intercorre fra il primo recupero dei dati e il momento in cui si chiama il metodo *Refresh*.

Abilitazione degli eventi mouse, tastiera e timer

La proprietà *Enabled* di un controllo dati determina se quest'ultimo risponde a eventi mouse, tastiera o timer e passa informazioni alla propria sorgente dati. L'impostazione predefinita di questa proprietà è **true**.

Per impedire che gli eventi mouse, tastiera o timer arrivino a un controllo dati, la proprietà *Enabled* deve essere impostata a **false**. Quando *Enabled* è **false**, il datasource che collega il controllo al proprio dataset non riceve informazioni dal controllo dati. Quest'ultimo continua a visualizzare i dati, ma il testo presentato nel controllo non può essere modificato.

Scelta del modo in cui organizzare i dati

Quando si costruisce l'interfaccia utente per l'applicazione database, si ha la possibilità di organizzare a proprio piacimento la visualizzazione delle informazioni e i controlli che trattano quelle informazioni.

Una delle prime decisioni da prendere è quella relativa al fatto che si voglia visualizzare un singolo record per volta o più record contemporaneamente.

Inoltre, si vorranno aggiungere dei controlli per spostarsi tra i record e per trattarli. Il controllo *TDBNavigator* fornisce un supporto nativo per la maggior parte delle operazioni che si potrebbero compiere.

Visualizzazione di un singolo record

In molte applicazioni, potrebbe essere opportuno fornire informazioni su un singolo record di dati per volta. Ad esempio, un'applicazione per l'immisione di ordini potrebbe visualizzare le informazioni relative a un singolo ordine senza indicare quali altri ordini sono attualmente registrati. Probabilmente queste informazioni provengono da un singolo record in un dataset di ordini.

Di solito, le applicazioni che visualizzano un singolo record risultano facili da leggere e da capire, perché tutte le informazioni del database sono relative allo stesso elemento (nel caso precedente, lo stesso ordine). I controlli data-aware in queste interfacce utente rappresentano un singolo campo di un record del database. La pagina Data Controls della Component palette fornisce un'ampia scelta di controlli per rappresentare vari tipi di campi. Di solito questi controlli sono versioni data-aware di altri controlli disponibili nella Component palette. Ad esempio, il controllo *TDBEdit* è una versione data-aware del controllo standard *TEdit* che consente agli utenti di visualizzare e modificare una stringa di testo.

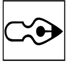
La scelta del controllo da utilizzare dipende dal tipo di dati (testo, testo formattato, immagini, informazioni booleane e così via) contenuti nel campo.

Visualizzazione dei dati come etichette

TDBText è un controllo a sola lettura simile al componente *TLabel* sulla pagina Standard della Component palette. Un controllo *TDBText* è utile quando si vogliono fornire dati unicamente da visualizzare su una scheda che consente l'immissione dati da parte dell'utente negli altri controlli. Si supponga, per esempio, di creare una scheda sui campi in una tabella di un elenco clienti e che, una volta che l'utente ha immesso le informazioni relative a un indirizzo, una città e uno stato o provincia nella scheda, si usi un consultazione dinamica per determinare automaticamente il campo codice postale da una tabella separata. Per visualizzare il campo relativo al codice postale corrispondente all'indirizzo immesso dall'utente, si potrebbe usare un componente *TDBText* legato alla tabella dei codici postali.

TDBText ricava il testo che visualizza da un campo specificato del record attivo di un dataset. Poiché *TDBText* preleva il testo da un dataset, il testo visualizzato è dinamico, ovvero cambia via via che l'utente si sposta nella tabella di database. Non è

pertanto possibile specificare il testo che *TDBText* deve visualizzare in fase di progetto, come avviene con *TLabel*.



Quando si inserisce in una scheda un componente *TDBText*, verificare che la relativa proprietà *AutoSize* sia **true** (impostazione predefinita) in modo che il controllo adatti automaticamente la propria dimensione per contenere dati di dimensione variabile. Se *AutoSize* è **false** e il controllo è troppo piccolo, i dati visualizzati vengono troncati.

Visualizzazione e modifica di campi in una casella di testo

TDBEdit è una versione associata ai dati di un componente casella di testo. *TDBEdit* visualizza il valore attivo di un campo dati al quale è collegato e ne consente la modifica con le tecniche standard delle caselle di testo.

Per esempio, si supponga che *CustomersSource* sia un componente *TDataSource* attivo, collegato a un componente *TClientDataSet* aperto, di nome *CustomersTable*. Un componente *TDBEdit* potrà essere inserito in una scheda e le sue proprietà potranno essere impostate come segue:

- *DataSource*: CustomersSource
- *DataField*: CustNo

Il componente casella di testo associato ai dati visualizza il valore della riga attiva della colonna CustNo del dataset *CustomersTable*, sia in fase di progetto sia in fase di esecuzione.

Visualizzazione e modifica del testo in un controllo memo l

TDBMemo è un componente associato ai dati, simile al componente standard *TMemo*-che può visualizzare dati di testo molto lunghi. *TDBMemo* visualizza il testo su più righe e consente a un utente di immettere testo su più righe. I controlli *TDBMemo* possono servire per visualizzare campi di testo molto lunghi e dati di testo contenuti in campi BLOB (Binary Large OBject).

Per impostazione predefinita, *TDBMemo* permette a un utente di modificare il testo del campo memo. Per impedire la modifica, impostare la proprietà *ReadOnly* del controllo memo a **true**. Per visualizzare i caratteri di tabulazione e permettere agli utenti di registrarli in un campo memo, impostare la proprietà *WantTabs* a **true**. Per limitare il numero di caratteri che l'utente può immettere nel campo memo del database, usare la proprietà *MaxLength*. Il valore predefinito per *MaxLength* è 0, che indica che non esiste alcun limite per i caratteri, tranne quello imposto dal sistema operativo.

Diverse proprietà determinano il modo in cui si presenta il memo del database e viene immesso il testo. Con la proprietà *ScrollBars* è possibile inserire barre di scorrimento nel memo. Per impedire il ritorno a capo automatico, impostare la proprietà *WordWrap* a **false**. La proprietà *Alignment* determina la modalità di allineamento del testo all'interno del controllo. Scelte possibili sono *taLeftJustify* (impostazione predefinita), *taCenter* e *taRightJustify*. Per cambiare il carattere del testo, utilizzare la proprietà *Font*.

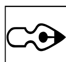
In fase di esecuzione gli utenti possono eseguire operazioni di taglio, copia e incollaggio di testo in un controllo memo di database. Le stesse operazioni possono

essere svolte da programma ricorrendo ai metodi *CutToClipboard*, *CopyToClipboard*, e *PasteFromClipboard*.

Poiché *TDBMemo* può visualizzare grossi volumi di dati, la visualizzazione completa durante l'esecuzione può richiedere del tempo. Per ridurre il tempo necessario per scorrere i record dei dati, *TDBMemo* dispone di una proprietà *AutoDisplay* che controlla se i dati ai quali si accede devono essere visualizzati automaticamente. Se si imposta *AutoDisplay* a **false**, invece dei dati effettivi *TDBMemo* visualizza il nome di campo. Per visualizzare i dati veri e propri, fare doppio clic all'interno del controllo.

Visualizzazione e modifica del testo in un controllo rich edit

TDBRichEdit è un componente associato ai dati-simile al componente standard *TRichEdit*-che può visualizzare del testo formattato memorizzato in un campo BLOB (binary large object). *TDBRichEdit* visualizza il testo formattato su più righe e consente all'utente di inserire del testo dello stesso tipo.



Benché *TDBRichEdit* fornisca proprietà e metodi per immettere dati e lavorare con un Rich Text, non fornisce nessun componente di interfaccia utente per rendere disponibili all'utente queste opzioni di configurazione. L'applicazione deve implementare l'interfaccia utente per rendere evidenti le capacità relative ai Rich Text.

Per impostazione predefinita, *TDBRichEdit* permette a un utente di modificare un testo di un campo memo. Per impedire la modifica, impostare la proprietà *ReadOnly* del controllo a **true**. Per visualizzare i caratteri di tabulazione e permettere agli utenti di registrarli in un campo memo, impostare la proprietà *WantTabs* a **true**. Per limitare il numero di caratteri che l'utente può immettere nel campo memo del database, usare la proprietà *MaxLength*. Il valore predefinito per MaxLength è 0, che indica che non esiste alcun limite per i caratteri, tranne quello imposto dal sistema operativo.

Poiché *TDBRichEdit* può visualizzare grossi volumi di dati, la visualizzazione completa durante l'esecuzione può richiedere del tempo. Per ridurre il tempo necessario a scorrere i record dei dati, *TDBRichEdit* dispone di una proprietà *AutoDisplay* che controlla se i dati ai quali si accede devono essere visualizzati automaticamente. Se si imposta *AutoDisplay* a **false**, invece dei dati effettivi *TDBRichEdit* visualizza il nome di campo. Per visualizzare i dati veri e propri, fare doppio clic all'interno del controllo.

Visualizzazione e modifica di campi grafici in un controllo immagine

TDBImage è un controllo associato ai dati che consente di visualizzare dati grafici contenuti in campi BLOB.

Per impostazione predefinita, *TDBImage* permette di modificare un'immagine con operazioni di taglia e incolla, dagli e negli Appunti, utilizzando i metodi e *CutToClipboard*, *CopyToClipboard* , e *PasteFromClipboard*. È possibile, invece, creare dei propri metodi per l'editing associati ai gestori di evento del controllo.

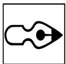
Per impostazione predefinita, un controllo immagine visualizza l'immagine fino a riempire completamente il controllo, ritagliando l'immagine nel caso sia troppo grande. È possibile impostare la proprietà *Stretch* a **true** per dimensionare il grafico in funzione della dimensione del controllo dell'immagine.

Poiché *TDBImage* può visualizzare quantità ingenti di dati, la visualizzazione completa durante l'esecuzione può richiedere del tempo. Per ridurre il tempo richiesto per effettuare lo scorrimento dei record di dati, *TDBImage* ha una proprietà *AutoDisplay* che determina se i dati ai quali si accede devono essere visualizzati automaticamente. Se *AutoDisplay* è impostata a **false**, *TDBImage* visualizza il nome del campo invece dei dati effettivi. Per visualizzare i dati veri e propri, fare doppio clic all'interno del controllo.

Visualizzazione e modifica di dati in caselle combinate e di riepilogo

Esistono quattro controlli dati che presentano all'utente, in fase di esecuzione, un insieme di valori predefiniti fra cui effettuare una scelta. I controlli seguenti sono le versioni data-aware dei controlli standard casella di riepilogo e casella combinata:

- *TDBListBox*, che visualizza una lista a scorrimento di elementi di cui l'utente ne può selezionare uno da immettere in un campo dati. Una casella di riepilogo data-aware visualizza un valore predefinito per un certo campo del record corrente e ne evidenzia la voce corrispondente nella lista. Se il valore del campo della riga corrente non è incluso nella lista, nella casella di riepilogo non viene evidenziato alcun valore. Quando un utente seleziona un elemento dell'elenco, il valore del campo corrispondente nel dataset sottostante viene modificato.
- *TDBComboBox*, che combina le funzionalità di un controllo data-aware per l'editing e di una lista a discesa. In fase di esecuzione può visualizzare un elenco a comparsa nel quale l'utente può scegliere uno dei valori predefiniti oltre a consentire l'immissione di un valore completamente diverso.
- *TDBLookupListBox*, che si comporta come *TDBListBox* tranne per il fatto che la lista degli elementi visualizzati viene cercata in un altro dataset.
- *TDBLookupComboBox*, che si comporta come *TDBComboBox* tranne per il fatto che la lista degli elementi visualizzati viene cercata in un altro dataset.



In fase di esecuzione gli utenti possono utilizzare la ricerca incrementale per trovare gli elementi di una casella di riepilogo. Quando il controllo ha il fuoco, ad esempio, inserendo 'ROB' si seleziona la prima voce della casella di riepilogo che inizia con le lettere 'ROB'. Se si immette un ulteriore carattere 'E', sarà selezionato il primo elemento che inizia con 'ROBE', ad esempio 'Robert Johnson'. La ricerca non distingue tra caratteri maiuscoli e minuscoli. *Backspace* ed *Esc* annullano la stringa di ricerca corrente (ma lasciano intatta la selezione), analogamente a una pausa di due secondi tra un tasto e l'altro.

Utilizzo di TDBListBox e TDBComboBox

Quando si utilizzano *TDBListBox* o *TDBComboBox*, è necessario utilizzare in progettazione lo String List editor per creare l'elenco degli elementi da visualizzare . Per visualizzare lo String List editor, fare clic nell'Object Inspector sul pulsante ellissi della proprietà *Items*.Quindi inserire nella lista gli elementi che si desidera visualizzare. In esecuzione, utilizzare i metodi della proprietà *Items* per trattare la lista di stringhe.

Quando un controllo *TDBListBox* o *TDBComboBox* è collegato con un campo tramite la proprietà *DataField*, il valore del campo viene visualizzato nella lista come

selezionato. Se il valore corrente non è nella lista, non viene visualizzato come selezionato alcun elemento. Tuttavia, *TDBComboBox* visualizza il valore corrente del campo nella casella di modifica, indipendentemente dal fatto che sia visualizzato o meno nella lista *Items*.

In *TDBListBox*, la proprietà *Height* determina quanti elementi sono visibili contemporaneamente nella casella di riepilogo. La proprietà *IntegralHeight* controlla il modo in cui potrà essere visualizzato l'ultimo elemento. Se *IntegralHeight* è **false** (impostazione predefinita), la parte inferiore della casella di riepilogo è determinata dalla proprietà *ItemHeight* e l'ultimo elemento potrebbe non essere visualizzato completamente. Se *IntegralHeight* è **true**, anche l'ultimo elemento della casella di riepilogo è visualizzato per intero.

In *TDBComboBox*, la proprietà *Style*determina interazione dell'utente con il controllo. Per impostazione predefinita, *Style* è *csDropDown*, il che significa che un utente può immettere valori da tastiera oppure scegliere un elemento dalla lista a discesa. Le seguenti proprietà determinano la modalità di visualizzazione dell'elenco *Items* in fase di esecuzione:

- *Style* determina lo stile di visualizzazione del componente:
  - *csDropDown* (impostazione predefinita): visualizza un elenco a comparsa con una casella di testo in cui l'utente può immettere un testo. Tutte le voci sono stringhe e hanno la stessa altezza.
  - *csSimple*: combina un casella di testo con un elenco di elementi aventi dimensioni fisse che viene sempre visualizzato. Quando si imposta *Style* a *csSimple*, accertarsi di aumentare la proprietà *Height* in modo che l'elenco venga visualizzato.
  - *csDropDownList*: visualizza un elenco a comparsa e una casella di testo, ma in fase di esecuzione l'utente non può immettere o modificare valori che non siano nell'elenco a comparsa.
  - *csOwnerDrawFixed* and *csOwnerDrawVariable*: consente all'elenco di elementi di visualizzare valori che non siano stringhe (per esempio, immagini bitmap) o di usare font differenti per i singoli elementi dell'elenco
- *DropDownCount*: indica il numero massimo di elementi che possono essere visualizzati nell'elenco. Se il numero di elementi contenuti in *Items* è maggiore di quello indicato in *DropDownCount*, l'utente può effettuare lo scorrimento dell'elenco. Se il numero di elementi contenuti in *Items* è minore di quello indicato in *DropDownCount*, l'elenco sarà sufficientemente grande per visualizzare tutte le voci contemporaneamente.
- *ItemHeight*: indica l'altezza di ciascun elemento quando lo stile è *csOwnerDrawFixed*.
- *Sorted*: se è **true**, gli elementi dell'elenco *Items* sono visualizzati in ordine alfabetico.

Visualizzazione e modifica di dati in liste e caselle combinate di consultazione

Le caselle di riepilogo di consultazione e le caselle combinate di consultazione (*TDBLookupListBox* e *TDBLookupComboBox*) presentano all'utente a una lista limitata

di scelte utilizzabili per assegnare al campo un valore valido. Quando un utente seleziona un elemento dell'elenco, il valore del campo corrispondente nel dataset sottostante viene modificato.

Ad esempio, si consideri una scheda per l'immissione di ordini, i cui campi siano collegati alla tabella *OrdersTable*. *OrdersTable* contiene un campo *CustNo* che corrisponde al codice identificativo di un cliente, ma non contiene altra informazione sui clienti. La tabella *CustomersTable* contiene, invece, un campo *CustNo* corrispondente a un ID cliente e riporta anche altre informazioni, quali l'azienda e l'indirizzo del cliente. Sarebbe molto pratico se la scheda degli ordini consentisse a un impiegato di selezionare un cliente in base al nome dell'azienda anziché mediante l'ID cliente quando viene creata una fattura. Un componente *TDBLookupListBox*, che visualizzi tutti i nomi delle aziende presenti in *CustomersTable*, permetterebbe all'utente di selezionare dalla lista il nome dell'azienda e di impostare correttamente il *CustNo* sul modulo d'ordine.

Questi controlli di consultazione estraggono un elenco di elementi da visualizzare da una delle due sorgenti indicate:

- **Un campo di consultazione definito per un dataset.**
  Per specificare le voci della casella di riepilogo utilizzando un campo di consultazione, il dataset al quale il controllo verrà collegato deve già definire un campo di consultazione. (Questo processo viene descritto in "Definizione di un campo di consultazione" a pagina 23-9). Per specificare il campo di consultazione per gli elementi della casella di riepilogo:

  1. Impostare la proprietà *DataSource* della casella di riepilogo al datasource del dataset contenente il campo di consultazione da usare.
  2. Scegliere il campo di consultazione da usare nell'elenco a comparsa della proprietà *DataField*.

  Quando si attiva una tabella associata a un controllo di consultazione, il controllo riconosce che il campo dati è un campo di consultazione e visualizza i valori appropriati tratti da quest'ultimo.

- **Un datasource secondario, un campo dati e una chiave**.
  Se non è stato definito un campo di ricerca per un dataset, è possibile stabilire una relazione analoga utilizzando una sorgente dati secondaria, un valore di campo in cui cercare nella sorgente dati secondaria e un valore di campo da restituire come elenco di elementi. Per specificare una sorgente dati secondaria per le voci della casella di riepilogo:

  3. Impostare la proprietà *DataSource* della casella di riepilogo alla sorgente dati del controllo.
  4. Scegliere nell'elenco a comparsa della proprietà *DataField*un campo in cui inserire i valori cercati. Il campo scelto non può essere un campo di consultazione.
  5. Impostare la proprietà *ListSource* della casella di riepilogo sulla sorgente dati del dataset contenente il campo in cui ricercare i valori.
  6. Scegliere nell'elenco a comparsa della proprietà *KeyField* un campo da utilizzare come chiave di consultazione. L'elenco a comparsa visualizza i campi per il

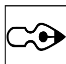
dataset associato alla sorgente dati specificata al punto 3. Il campo scelto non deve necessariamente appartenere a un indice ma, se vi appartiene, la ricerca sarà ancora più rapida.

7 Scegliere dall'elenco a comparsa della proprietà *ListField*un campo di cui restituire i valori. L'elenco a comparsa visualizza i campi del dataset associato alla sorgente dati specificata al punto 3.

Quando si attiva una tabella associata a un controllo di consultazione, il controllo riconosce che le voci del suo elenco a comparsa sono tratte da una sorgente secondaria e visualizza i valori appropriati tratti da quest'ultima.

Per specificare il numero di elementi visualizzati in un controllo *TDBLookupListBox* in un determinato momento, utilizzare la proprietà *RowCount*. L'altezza della casella di riepilogo viene adattata per contenere esattamente il numero di righe specificato.

Per specificare il numero di elementi visualizzati nella lista a discesa di *TDBLookupComboBox*, utilizzare invece la proprietà *DropDownRows*.

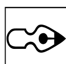


È possibile anche impostare una colonna in una griglia di dati in modo che agisca come una casella combinata di consultazione. Per informazioni sull'operazione, consultare "Definizione di una colonna elenco di consultazione" a pagina 19-23.

Gestione dei valori di campo booleani con le caselle di controllo

TDBCheckBox è un controllo casella di controllo associata ai dati. Può essere usato per impostare i valori di campi boolean di un dataset. Una scheda di fattura cliente, ad esempio, potrebbe avere un controllo casella di controllo che, se attivato, indica che il cliente non è soggetto a tassazione, e se disattivato che il cliente deve pagare la suddetta tassa.

Il controllo casella di controllo data-aware gestisce il proprio stato di attivo o inattivo confrontando il valore del campo attuale con i contenuti e le proprietà *ValueChecked* e *ValueUnchecked*. Se il valore del campo coincide con la proprietà *ValueChecked*, il controllo è selezionato. Altrimenti, se il campo coincide con la proprietà *ValueUnchecked*, il controllo non è selezionato.



I valori *ValueChecked* e *ValueUnchecked* non possono essere identici.

Impostare la proprietà *ValueChecked* a un valore che il controllo, se attivato, deve inserire nel database quando l'utente si sposta in un altro record. Per impostazione predefinita, questo valore è impostato a "true", ma è possibile dargli qualsiasi valore alfanumerico adatto alle proprie esigenze. Mentre il valore di *ValueChecked* è attivo, è anche possibile registrare un elenco di elementi separati con punto e virgola. Se qualche elemento del record attuale coincide con i contenuti di quel campo, la casella di controllo viene attivata. Si può, ad esempio, specificare una stringa *ValueChecked* come segue:

```
DBCheckBox1->ValueChecked = "true;Yes;On";
```

Se il campo del record attuale contiene valori di "True", "Yes" o "On" la casella di controllo viene attivata. Il confronto del campo con le stringhe *ValueChecked* non fa differenza fra maiuscole e minuscole. Se un utente controlla una casella per la quale ci sono stringhe *ValueChecked* multiple, la prima stringa è il valore che viene assegnato al database.

Se il controllo non è attivo quando l'utente si sposta su un altro record, occorre impostare la proprietà *ValueUnchecked* al valore che il controllo dovrebbe assegnare al database. Per impostazione predefinita, questo valore è impostato a “False”, ma è possibile dargli un qualsiasi valore alfanumerico adatto alle proprie esigenze. Mentre il valore di *ValueChecked* è attivo, è anche possibile registrare un elenco di elementi separati con punto e virgola. Se nessun elemento del record attuale coincide con i contenuti di quel campo, la casella di controllo risulta deselezionata.

Una casella di controllo associata ai dati viene disattivata ogni volta che il campo del record attuale non contiene uno dei valori elencati nelle proprietà *ValueChecked* e *ValueUnchecked*.

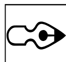
Se il campo a cui è associata una casella di controllo è un campo logico, la casella di controllo controlla sempre se i contenuti del campo sono **true**, e si disattiva se i contenuti del campo sono **false**. In questo caso, le stringhe immesse nelle proprietà *ValueChecked* e *ValueUnchecked* non hanno effetto sui campi logici.

Restrizione dei valori del campo con controlli di opzione

TDBRadioGroup è una versione associata ai dati di un controllo gruppo di opzioni. Consente di impostare il valore di un campo dati con un controllo pulsante di opzione, in cui esiste un numero limitato di possibili valori del campo. Il gruppo di opzioni è composto da un pulsante per ciascuno valore che il campo può accettare. Gli utenti possono impostare il valore di un campo dati selezionando il pulsante di opzione desiderato.

La proprietà *Items* determina i pulsanti di opzione che appaiono nel gruppo. *Items* è una lista di stringhe. Per ciascuna stringa di *Items*, viene visualizzato un pulsante di opzione e ciascuna stringa appare alla destra di un pulsante di opzione sotto forma di etichetta del pulsante.

Se il valore corrente di un campo associato con un gruppo di opzioni coincide con una delle stringhe nella proprietà *Items*, il relativo pulsante di opzione appare selezionato. Se, ad esempio, *Items* elenca tre stringhe, “Rosso”, “Giallo” e “Blu”, e il campo del record attivo contiene il valore “Blu”, il terzo pulsante del gruppo viene selezionato.



Se il campo non coincide con nessuna stringa di *Items*, è ancora possibile selezionare un pulsante di opzione a patto che il campo coincida con una stringa della proprietà *Values*. Se il campo del record attivo non coincide con nessuna stringa di *Items* o *Values*, non viene selezionato alcun pulsante di opzione.

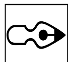
La proprietà *Values* può contenere un elenco facoltativo di stringhe che possono essere restituite al dataset quando un utente seleziona un pulsante di opzione e assegna un record. Le stringhe vengono associate con i pulsanti in sequenza numerica. La prima stringa è associata al primo pulsante, la seconda al secondo pulsante e così via. Supponiamo, ad esempio, che *Items* contenga “Rosso”, “Giallo” e “Blu”, e che Values contenga “Magenta”, “Giallo” e “Azzurro”. Se un utente seleziona il pulsante identificato come “Rosso”, il valore “Magenta” viene inserito nel database.

Se non si forniscono stringhe di *Values*, quando un record viene assegnato, la stringa di *Items* di un pulsante di opzione selezionato viene inserita nel database.

Visualizzazione di più record

Talvolta potrebbe essere utile visualizzare molti record nella stessa scheda. Ad esempio, un'applicazione per la fatturazione potrebbe visualizzare tutti gli ordini fatti da un singolo cliente sulla stessa scheda.

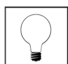
Per visualizzare più record, utilizzare un controllo griglia. I controlli griglia offrono una vista multi campo e multi record dei dati che può rendere l'interfaccia utente dell'applicazione più accattivante ed efficace. Tali controlli sono trattati nei paragrafi "Visualizzazione e modifica dei dati con TDBGrid" a pagina 19-16 e "Creazione di una griglia contenente altri controlli associati ai dati" a pagina 19-30.



È impossibile visualizzare più record quando si utilizza un dataset unidirezionale.

Si potrebbe voler progettare un'interfaccia utente che visualizzi sia i campi di un singolo record sia le griglie per rappresentare più record. Esistono sono due modelli che combinano questi due approcci:

- **Schede master-detail:** È possibile rappresentare le informazioni provenienti da una tabella principale e da una tabella di dettaglio includendo sia controlli per visualizzare un singolo campo sia controlli griglia. Ad esempio, si potrebbero visualizzare le informazioni relative a un singolo cliente con una griglia di dettaglio che visualizzi gli ordini di quel cliente. Per informazioni sul collegamento delle tabelle sottostanti in una scheda master-detail, consultare i paragrafi "Creazione di relazioni master/detail" a pagina 22-36 e "Impostazione di relazioni master/detail mediante parametri" a pagina 22-49.
- **Schede drill-down**: In una scheda che visualizza più record, è possibile includere controlli per singolo campo che visualizzano informazioni dettagliate solo dal record corrente. Questo approccio è particolarmente utile quando i record includono lunghi memo o informazioni grafiche. Man mano che l'utente scorre i record della griglia, il campo memo o il grafico vengono aggiornati in modo da rappresentare il valore del record corrente. L'impostazione di questo schema è molto semplice. La sincronizzazione tra le due visualizzazioni è automatica se la griglia e il controllo per il campo memo o per l'immagine condividono un datasource comune.



Normalmente, la combinazione di questi due approcci su una singola scheda non è una buona idea. In tali schede risulta solitamente difficile capire per l'utente quali sono le relazioni che intercorrono tra i dati.

Visualizzazione e modifica dei dati con TDBGrid

Un controllo *TDBGrid* consente di visualizzare e di modificare i record da un dataset in un formato griglia tabellare.

Figura 19.1 Controllo TDBGrid

Campo corrente

Titoli di colonna

Indicatore di record

VendorName	Address1	City	State
Cacor Corporation	161 Southfield Rd	Southfield	OH
Underwater	50 N 3rd Street	Indianapolis	IN
J.W. Luscher Mfg.	65 Addams Street	Berkely	MA
Scuba Professionals	3105 East Brace	Rancho Dominguez	CA
Divers' Supply Shop	5208 University Dr	Macon	GA
Techniques	52 Dolphin Drive	Redwood City	CA
Perry Scuba	3443 James Ave	Hapeville	GA

Questi tre fattori incidono sull'aspetto dei record visualizzati in un controllo griglia:

- Esistenza di oggetti colonna persistente definiti per la griglia utilizzando Columns editor. Gli oggetti colonna persistente forniscono una grande flessibilità nell'impostazione dell'aspetto della griglia e dei dati. Per informazioni sull'uso di colonne persistenti, consultare"Creazione di una griglia personalizzata" a pagina 19-18.
- Creazione di componenti campo persistente per il dataset visualizzato nella griglia. Per ulteriori informazioni sulla creazione di componenti campo persistente usando Fields editor, consultare Capitolo 23, "Operazioni con i componenti campo".
- Impostazione della proprietà *ObjectView* del dataset per le griglie che visualizzano i campi ADT e i campi array. Consultare "Visualizzazione dei campi ADT e array" a pagina 19-24.

Un controllo griglia ha una proprietà *Columns* che è di per sé un wrapper in un oggetto *TDBGridColumns*. *TDBGridColumns* è una raccolta di oggetti *TColumn* che rappresentano tutte le colonne di un controllo griglia.Per impostare, in fase di progettazione, gli attributi delle colonne, si può ricorrere all'editor Columns, oppure si può usare la proprietà *Columns* della griglia per accedere alle proprietà, agli eventi e ai metodi di *TDBGridColumns* in fase di esecuzione.

Uso di un controllo griglia nel suo stato predefinito

La proprietà *State* della proprietà *Columns* di una griglia indica se esiste l'oggetto colonna persistente per la griglia. *Columns->State* è una proprietà di sola esecuzione che viene automaticamente impostata per una griglia. Lo stato predefinito è *csDefault*, che indica che non esistono oggetti colonna persistente per la griglia. In questo caso la visualizzazione dei dati nella griglia è determinata principalmente dalle proprietà dei campi nel dataset della griglia, oppure, nel caso non vi siano componenti campo persistenti, da un insieme predefinito di caratteristiche di visualizzazione.

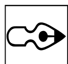
Quando la proprietà *Columns->State* di una griglia è *csDefault*, le colonne della griglia vengono generate dinamicamente dai campi visibili del dataset, e l'ordine delle colonne nella griglia deve corrispondere a quello dei campi del dataset. Ogni colonna

della griglia viene associata a un componente field. Le modifiche delle proprietà apportate ai componenti campo appaiono immediatamente nella griglia.

L'uso di un controllo griglia con colonne generate dinamicamente è utile per la visualizzazione e la modifica del contenuto di tabelle arbitrarie selezionate in fase di esecuzione. Dal momento che la struttura della griglia non è definita, può cambiare in modo dinamico per adattarsi ai diversi database. Una singola griglia con colonne generate dinamicamente può visualizzare ora una tabella Paradox, poi il risultato di una query SQL quando la proprietà *DataSource* della griglia cambia o quando viene modificata la proprietà *DataSet* dello stesso datasource.

È possibile cambiare l'aspetto di una colonna dinamica in fase di progettazione o di esecuzione, ma ciò che viene di fatto modificato sono le proprietà corrispondenti del componente campo visualizzato nella colonna. Le proprietà delle colonne dinamiche esistono soltanto finché una colonna viene associata con uno specifico campo in un singolo dataset. Per esempio, la modifica della proprietà *Width* di una colonna incide sulla proprietà *DisplayWidth* del campo associato a quella colonna. Le modifiche apportate alle proprietà delle colonne che non si basano sulle proprietà dei campi, come *Font*, esistono solo per la durata della colonna.

Se il dataset di una griglia è costituito da componenti campo dinamici, i campi vengono distrutti ogni volta che il dataset viene chiuso. Quando i componenti campo sono distrutti, tutte le colonne dinamiche ad essi associate vengono distrutte a loro volta.Se il dataset di una griglia è costituito da componenti campo persistente, i componenti campo continuano ad esistere anche dopo che il dataset viene chiuso, cosicché le colonne associate a tali campi mantengono le loro proprietà quando il dataset viene chiuso.



La modifica della proprietà *Columns->State* di una griglia in *csDefault* in fase di esecuzione cancella tutti gli oggetti colonna della griglia (anche le colonne persistenti) e ricostruisce le colonne dinamiche sulla base dei campi visibili del dataset della griglia.

Creazione di una griglia personalizzata

Una griglia personalizzata è quella per il quale si definiscono oggetti colonna persistente che descrivono come appare una colonna e come vi vengono visualizzati i dati. Una griglia personalizzata consente di configurare diverse griglie in modo da presentare viste differenti dello stesso dataset (per esempio, ordinamenti di colonna diversi, opzioni di campo diverse e colori e font per le colonne differenti). Una griglia personalizzata, inoltre, consente agli utenti di modificare l'aspetto della griglia in fase di esecuzione senza influire sui campi utilizzati dalla griglia o sull'ordinamento dei campi del dataset.

Le griglie personalizzate devono essere usate preferibilmente con i dataset la cui struttura è nota in fase di progettazione. Dal momento che esse si aspettano che nel dataset esistano i nomi di campo stabiliti in fase di progettazione, le griglia personalizzate sono poco adatte allo scorrimento di tabelle arbitrarie selezionate in fase di esecuzione.

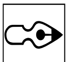
Caratteristiche delle colonne persistenti

Quando si creano oggetti colonna persistente per una griglia, esse vengono solo debolmente associate con i campi sottostanti nel dataset della griglia. I valori predefiniti delle proprietà delle colonne persistenti vengono derivati in modo dinamico da una sorgente predefinita (il campo associato o la stessa griglia) finché non viene assegnato un valore alla proprietà della colonna. Finché non si effettua questa operazione, il valore della proprietà della colonna cambia via via che cambia la sua sorgente predefinita. Una volta assegnato un valore alla proprietà di una colonna, tale valore non cambia più anche se cambia la sua sorgente predefinita.

Per esempio, la sorgente predefinita per il titolo di una colonna è la proprietà *DisplayLabel* di un campo associato. Se si modifica la proprietà *DisplayLabel*, il titolo della colonna riflette immediatamente la modifica. Se si assegna una stringa ai titoli delle colonne, questi diventano indipendenti dalla proprietà *DisplayLabel* dei campi associati. Le modifiche successive apportate alla proprietà *DisplayLabel* del campo non incidono più sul titolo della colonna.

Le colonne persistenti esistono indipendentemente dai componenti campo ai quali sono associati. Infatti, le colonne persistenti non devono essere associate con gli oggetti campo. Se la proprietà *FieldName* di una colonna persistente è vuota, o se il nome del campo non corrisponde a quello di nessun campo del dataset attivo della griglia, la proprietà *Field* della colonna è NULL e la colonna viene tracciata con celle vuote. Se si ridefinisce il metodo predefinito per il disegno della cella, nelle celle vuote è possibile visualizzare informazioni personalizzate. Ad esempio, è possibile utilizzare una colonna vuota per visualizzare sull'ultimo record di un gruppo di record i valori di riepilogo del gruppo. Un'altra possibilità potrebbe essere quella di visualizzare una bitmap o un grafico a barre che rappresenti graficamente un certo aspetto dei dati del record.

È possibile associare due o più colonne persistenti allo stesso campo di un dataset. Per esempio, si può visualizzare un campo numero di parte alle estremità sinistra e destra di una grande griglia per consentire di trovarlo più facilmente senza dover scorrere la griglia.



Dal momento che le colonne persistenti non debbono essere associate a un campo del dataset, e che più colonne possono far riferimento allo stesso campo, la proprietà *FieldCount* di una griglia personalizzata può avere un valore inferiore o uguale a quello della colonna della griglia. Inoltre, si tenga presente che, se la colonna al momento selezionata in una griglia personalizzata non è associata a un campo, la proprietà *SelectedField* della griglia è NULL e la proprietà *SelectedIndex* è -1.

Le colonne persistenti possono essere configurate per visualizzare le celle delle griglie come una casella combinata con un elenco a comparsa di valori di consultazione provenienti da un altro dataset o da un elenco statico di opzioni, o come un pulsante ellissi (...) in una cella su cui è possibile fare clic per attivare degli speciali visualizzatori di dati o delle finestre di dialogo correlate alla cella attiva.

Creazione di colonne persistenti

Per personalizzare l'aspetto della griglia in fase di progettazione, si deve chiamare l'editor Columns per creare un gruppo di oggetti colonna persistente per la griglia. In

fase di esecuzione, la proprietà *State* di una griglia con oggetti colonna persistente viene impostata automaticamente a *csCustomized*.

Per creare colonne persistenti per un controllo griglia:

1 Selezionare il componente griglia nella scheda.
2 Richiamare Columns editor facendo doppio clic nell'Object Inspector sulla proprietà *Columns* della griglia.

La casella di selezione Columns visualizza le colonne persistenti che sono state definite per la griglia selezionata. Quando si richiama per la prima volta l'editor Columns, questo elenco è vuoto in quanto la griglia è nel suo stato predefinito e contiene quindi solo colonne dinamiche.

Si possono creare contemporaneamente colonne persistenti per tutti i campi di un dataset, oppure si possono creare singole colonne persistenti. Per creare colonne persistenti per tutti i campi:

1 Fare clic destro sulla griglia per richiamare il menu contestuale e scegliere Add All Fields. Si noti che, se la griglia non è già associata a un datasource, Add All Fields è disattivato. Prima di scegliere Add All Fields, occorre associare la griglia a un datasource che abbia un dataset attivo.
2 Se la griglia contiene già colonne persistenti, comparirà un finestra di dialogo che chiederà se si vogliono eliminare le colonne esistenti, o se si intende aggiungere le nuove colonne al gruppo esistente. Se si sceglie Yes, tutte le informazioni sulle colonne persistenti esistenti vengono rimosse, e tutti i campi del dataset attivo vengono inseriti per nome di campo secondo il relativo ordine nel dataset. Se si sceglie No, tutte le informazioni sulle colonne persistenti esistenti vengono mantenute, e le informazioni sulle nuove colonne, basate sui campi aggiuntivi del dataset, vengono aggiunte al dataset.
3 Fare clic su Close per applicare le colonne persistenti alla griglia e chiudere la finestra di dialogo.

Per creare singole colonne persistenti:

1 Scegliere il pulsante Add nel Columns editor. La nuova colonna verrà selezionata nella casella di selezione. Alla nuova colonna vengono attribuiti un numero sequenziale e un nome predefinito (per esempio, 0 - TColumn).
2 Per associare un campo a questa nuova colonna, impostare la proprietà *FieldName* nell'Object Inspector.
3 Per impostare il titolo della nuova colonna, espandere la proprietà *Title* nell'Object Inspector e impostare la proprietà *Caption*.
4 Chiudere Columns editor per applicare le colonne persistenti alla griglia e chiudere la finestra di dialogo.

In fase di esecuzione, è possibile passare a colonne persistenti assegnando *csCustomized* alla proprietà *Columns*::*State*. Tutte le colonne della griglia vengono distrutte e vengono costruite nuove colonne persistenti per ciascun campo del dataset della griglia. In esecuzione sarà possibile poi aggiungere una colonna persistente chiamando il metodo *Add* della lista di colonne

```
DBGrid1->Columns->Add();
```

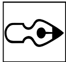
Cancellazione di colonne persistenti

La cancellazione di una colonna persistente da una griglia risulta utile per eliminare i campi che non devono essere visualizzati. Per rimuovere una colonna persistente da una griglia:

1 Fare doppio clic sulla griglia per visualizzare il Columns editor.

2 Selezionare il campo da rimuovere nella casella di selezione Columns.

3 Fare clic su Delete (in alternativa, per rimuovere una colonna si può anche ricorrere al menu contestuale o al tasto ***Canc***).



Se si cancellano tutte le colonne di una griglia, la proprietà *Columns->State* ritorna al suo stato *csDefault* e costruisce automaticamente colonne dinamiche per ogni campo del dataset.

In esecuzione è possibile cancellare una colonna persistente semplicemente liberando l'oggetto colonna:

```
delete DBGrid1->Columns->Items[5];
```

Sistemazione dell'ordine delle colonne persistenti

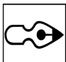
L'ordine in cui le colonne vengono visualizzate nell'editor Columns è lo stesso con cui appaiono nella griglia. Per cambiare tale ordine, basta trascinare e rilasciare le colonne all'interno della casella di selezione Columns.

Per cambiare l'ordine di una colonna:

1 Selezionare la colonna nella casella di selezione Columns.

2 Trascinarla in una nuova posizione nella casella di selezione.

È anche possibile cambiare l'ordine delle colonne facendo clic sul titolo della colonna e trascinandola nella nuova posizione.



Il riordinamento dei campi persistenti nel Fields editor riordina anche le colonne in una griglia predefinita, ma non in una griglia personalizzata.



Non è possibile riordinare in fase di progettazione le colonne nelle griglie che contengono colonne e campi dinamici, dal momento che non esiste alcunché di persistente per registrare il campo modificato o l'ordine delle colonne.

In fase di esecuzione, se la proprietà *DragMode*della griglia è impostata a *dmManual*, l'utente può utilizzare il mouse per trascinare una colonna in una nuova posizione all'interno della griglia. Il riordino delle colonne di una griglia, la cui proprietà *State* è impostata allo stato *csDefault*, riordina anche componenti campo nel dataset sottostante alla griglia. L'ordine dei campi nella tabella fisica resta inalterato. Per impedire all'utente di riordinare le colonne in fase di esecuzione, impostare la proprietà *DragMode* della griglia a *dmAutomatic*.

In esecuzione, lo spostamento di una colonna attiva l'evento *OnColumnMoved*della griglia.

Impostazione delle proprietà di una colonna in progettazione

Le proprietà di una colonna determinano il modo in cui i dati vengono visualizzati nelle relative celle. La maggior parte delle proprietà di una colonna ottiene i propri valori predefiniti dalle proprietà associate a un altro componente, detto *sorgente predefinita*, come ad esempio una griglia o un componente field associato

Per impostare le proprietà di una colonna, selezionare la colonna nel Columns editor e impostarne le proprietà nell'Object Inspector. La tabella seguente riporta le proprietà fondamentali di una colonna che è possibile impostare.

Tabella 19.2 Proprietà di Columns

Proprietà	Funzione
Alignment	Allinea a sinistra, a destra o al centro i dati del campo nella colonna. Sorgente predefinita: *TField*::*Alignment*.
ButtonStyle	*cbsAuto*: (impostazione predefinita) Visualizza un elenco a comparsa se il campo associato è un campo di consultazione, o se la proprietà PickList della colonna contiene dati. *cbsEllipsis*: Visualizza un pulsante con puntini di sospensione (...) a destra della cella. Facendo clic sul pulsante, si attiva l'evento *OnEditButtonClick* della griglia. *cbsNone*: La colonna usa solo il controllo normale di editing per modificare i dati della colonna.
Color	Specifica il colore di sfondo delle celle della colonna. Sorgente predefinita: *TDBGrid*::*Color*. (Per il colore di primo piano del testo, vedere la proprietà Font.)
DropDownRows	Numero di righe di testo visualizzate dall'elenco a comparsa. Impostazione predefinita: 7.
Expanded	Specifica se la colonna è espansa. Si applica solo alle colonne che rappresentano campi ADT o array.
FieldName	Specifica il nome del campo associato a questa colonna. Può essere vuoto.
ReadOnly	**true**: I dati della colonna non possono essere modificati dall'utente. **false**: (impostazione predefinita) I dati della colonna possono essere modificati.
Width	Specifica la larghezza della colonna in pixel sullo schermo. Sorgente predefinita: derivato da *TField*::*DisplayWidth*.
Font	Specifica il tipo di font, le dimensioni e il colore usati per rappresentare il testo nella colonna. Sorgente predefinita: *TDBGrid*::*Font*.
PickList	Contiene un elenco di valori da visualizzare in un elenco a comparsa nella colonna.
Title	Imposta le proprietà del titolo della colonna selezionata.

La tabella seguente riporta le opzioni che si possono specificare per la proprietà *Title*.

Tabella 19.3 Proprietà Title del TColumn espanso

Proprietà	Funzione
Alignment	Allinea a sinistra (impostazione predefinita), a destra o al centro il testo del titolo della colonna.
Caption	Specifica il testo da visualizzare nella colonna del titolo. Sorgente predefinita: *TField*::*DisplayLabel*.
Color	Specifica il colore di sfondo usato per tracciare la celle del titolo della colonna. Sorgente predefinita: *TDBGrid*::*FixedColor*.
Font	Specifica il tipo di font, le dimensioni e il colore usati per rappresentare il testo nella colonna. Sorgente predefinita: *TDBGrid*::*TitleFont*.

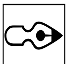
Definizione di una colonna elenco di consultazione

È possibile creare una colonna che visualizza una lista a discesa di valori, simili a un controllo casella combinata di consultazione. Per specificare che la colonna agisce come una casella combinata, impostare la proprietà *ButtonStyle* della colonna a *cbsAuto*. Una volta riempita la lista con valori, quando una cella di quella colonna è in modalità editing, la griglia visualizza automaticamente un pulsante simile a quello di una casella combinata a discesa.

Esistono due modi per riempire la lista con valori selezionabili dall'utente:

- È possibile ottenere i valori da una tabella di consultazione. Per fare in modo che una colonna visualizzi una lista a discesa di valori prelevati da una tabella di consultazione differente, è necessario definire nel dataset un campo di consultazione. Per informazioni sulla creazione di campi di consultazione, vedere il paragrafo "Definizione di un campo di consultazione" a pagina 23-9. Una volta definito il campo di consultazione, occorre impostare *FieldName* della colonna al nome di tale campo. La lista a comparsa viene riempita automaticamente con i valori di consultazione definiti tramite il campo di consultazione
- È possibile specificare esplicitamente in progettazione una lista di valori. Per immettere i valori della lista in fase di progettazione, fare doppio clic nell'Object Inspector sulla proprietà *PickList*della colonna. In questo modo viene attivato l'editor String List, in cui si immetteranno i valori che compongono la lista di scelte della colonna.

Per impostazione predefinita, la lista a discesa visualizza 7 valori. È possibile modificare la lunghezza di questa lista impostando la proprietà *DropDownRows*.



Per ripristinare il normale comportamento di una colonna a cui è stata assegnata una lista esplicita di scelte, cancellare tutto il testo dalla lista di scelte utilizzando lo String List editor.

Inserimento di un pulsante in una colonna

Una colonna può visualizzare un pulsante ellissi (...) a destra del normale editor di celle. *Ctrl+Invio* o un clic del mouse attivano l'evento *OnEditButtonClick* della griglia. Si può usare il pulsante ellissi per richiamare schede contenenti viste più dettagliate dei dati della colonna. Per esempio, in una tabella che visualizza prospetti di fatture, si

può impostare un pulsante ellissi nella colonna Totale fattura per richiamare una scheda che visualizzi le voci di quella fattura, o la formula usata per il calcolo dell'IVA, e così via. Per i campi grafici si può usare il pulsante ellissi per richiamare una scheda che visualizzi un'immagine

Per creare un pulsante ellissi in una colonna:

1 Selezionare la colonna nella casella di selezione *Columns*.
2 Impostare *ButtonStyle* a *cbsEllipsis*.
3 Scrivere un gestore di eventi *OnEditButtonClick*

Ripristino dei valori predefiniti in una colonna

In esecuzione è possibile controllare la proprietà *AssignedValues* di una colonna per determinare se una proprietà della colonna è stata assegnata esplicitamente. I valori che non sono stati definiti in modo esplicito derivano dinamicamente dal campo associato o dalle impostazioni predefinite della griglia.

È possibile annullare le modifiche apportate alle proprietà in una o più colonne. È sufficiente selezionare nel Columns editor la colonna o le colonne da ripristinare, quindi selezionare Restore Defaults dal menu contestuale. Restore Defaults annulla le impostazioni assegnate alle proprietà e riporta le proprietà di una colonna ai valori derivati dal relativo componente field sottostante

In esecuzione, è possibile reimpostare tutte le proprietà predefinite di una singola colonna chiamando il metodo *RestoreDefaults* della colonna. Chiamando il metodo *RestoreDefaults* dell'elenco di colonne, è inoltre possibile ripristinare le proprietà predefinite per tutte le colonne di una griglia:

```
DBGrid1->Columns->RestoreDefaults();
```

Visualizzazione dei campi ADT e array

Talvolta i campi del dataset della griglia non rappresentano valori semplici come testo, grafica, valori numerici, e così via. Alcuni server di database consentono campi che sono un composto di tipi di dati più semplici, come i campi ADT o campi di array.

Una griglia può visualizzare i campi compositi in due modi:

- Può " scomporre" il campo in modo che ognuno dei tipi più semplici che costituiscono il campo appaia come un campo separato nel dataset. Quando un campo composito viene scomposto, i suoi costituenti vengono visualizzati come campi separati, e la provenienza da una sorgente comune è messa in evidenza solo dal fatto che ogni nome di campo è preceduto dal nome del campo genitore comune incluso nella tabella del database sottostante.

  Per visualizzare i campi compositi come se fossero campi separati, impostare la proprietà *ObjectView* del dataset a **false**. Il dataset memorizza i campi compositi come un insieme di campi separati e la griglia si comporta di conseguenza assegnando a ogni singolo costituente una colonna separata.

- Può visualizzare campi compositi in una singola colonna, riflettendo il fatto che essi sono un singolo campo. Quando i campi compositi vengono visualizzati utilizzando una singola colonna, la colonna può essere espansa e compressa facendo clic sulla freccia presente nella barra del titolo del campo oppure impostando la proprietà *Expanded* della colonna:
  - Quando una colonna viene espansa, ogni campo figlio viene visualizzato nella propria sottocolonna, visualizzando, sotto la barra del titolo del campo genitore, una barra del titolo. La barra del titolo della griglia viene incrementata in altezza e riporta nella prima riga il nome del campo composito, mentre la seconda riga viene suddivisa tra le singole parti. La barra del titolo dei campi non composti risulta pertanto straordinariamente alta. Questa espansione continua per quei costituenti che, a loro volta, sono campi compositi (ad esempio, un tabella di dettaglio annidata in un'altra tabella di dettaglio), incrementando di conseguenza l'altezza della barra del titolo.
  - Quando il campo viene compresso, viene visualizzata solo una colonna contenente una stringa non modificabile, delimitata da virgole, che contiene i campi figlio.

  Per visualizzare un campo composito in una colonna espandibile e comprimibile, impostare la proprietà *ObjectView* del dataset a **true**. Il dataset memorizza il campo composito come un singolo componente campo che contiene un insieme di sottocampi annidati. La griglia riflette questa impostazione visualizzando una colonna che può essere espansa e compressa.

La Figura 19.2 mostra la griglia con un campo ADT e con un campo array. La proprietà ObjectView del dataset è impostata a false, cosicché ogni campo figlio ha una colonna.

Figura 19.2 Controllo TDBGrid con ObjectView impostato a false

Campi figlio ADT   Campi array figlio

ID_KEY	NAME_ADT.FIRST	NAME_ADT.MIDDLE	NAME_ADT.LAST	TELNOS_ARRAY[0]	TELNOS_ARRAY[1]	TELNOS_ARR/
1	Stephan		Wright	415-908-9875	902-786-1245	
2	Whitney	N	Long			
3	Chris	T	Scanlan	234-232-1343		

Le Figura 19.3 e 19.4 mostrano la griglia con un campo ADT e con un campo array. La Figura 19.3 mostra i campi ridotti. In questo stato non è possibile modificarli. La Figura 19.4 mostra i campi espansi. Per espandere e per ridurre i campi, occorre farvi sopra clic nella barra del titolo dei campi.

Figura 19.3 Controllo TDBGrid con Expanded impostato a false

ID_KEY	NAME_ADT	TELNOS_ARRAY
1	(Stephan, , Wright)	(415-908-9875, 902-786-1245, , , , , , , ,)
2	(Whitney, N, Long)	(, , , 510-454-7234, , , , , ,)
3	(Chris, T, Scanlan)	(234-232-1343, , , , , , , , ,)

Figura 19.4 Controllo TDBGrid con Expanded impostato a true

Colonne del campo figlio ADT

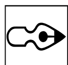
Colonne del campo array figlio

ID_KEY	NAME_ADT			TELNOS_ARRAY			
	FIRST	MIDDLE	LAST	TELNOS_ARRAY[0]	TELNOS_ARRAY[1]	TELNOS_ARRAY[2]	TELNO
1	Stephan		Wright	415-908-9875	902-786-1245		
2	Whitney	N	Long				510-454
3	Chris	T	Scanlan	234-232-1343			

La tabella seguente elenca le proprietà che incidono sul modo in cui appaiono i campi ADT e gli array in un *TDBGrid*:

Tabella 19.4 Proprietà che influiscono sulla modalità di visualizzazione dei campi compositi

Proprietà	Oggetto	Funzione
Expandable	TColumn	Indica se la colonna può essere espansa in modo da mostrare i campi figlio in colonne separate e modificabili. (a sola lettura)
Expanded	TColumn	Specifica se la colonna è espansa.
MaxTitleRows	TDBGrid	Specifica il numero massimo di righe di titolo che è possibile visualizzare nella griglia
ObjectView	TDataSet	Specifica se i campi sono visualizzati come “scomposti” o in modo oggetto, in cui ogni campo oggetto può essere espanso e compresso.
ParentColumn	TColumn	Fa riferimento all’oggetto TColumn a cui appartiene la colonna del campo figlio.



Oltre ai campi di tipo ADT, alcuni dataset includono campi che fanno riferimento a un altro dataset (campi dataset) o a un record in altri campi dataset (reference). Le griglie data-aware visualizzano rispettivamente tali campi come “(DataSet)” o “(Reference)”. In fase di esecuzione, alla destra di tali campi appare un pulsante con ellissi. Facendo clic sul pulsante con ellissi viene attivata una nuova scheda con una griglia che visualizza il contenuto del campo. Per i campi dataset, questa griglia visualizza il dataset che rappresenta il valore del campo. Per campi di riferimento, questa griglia contiene una singola riga che visualizza il record di un altro dataset.

Impostazione delle opzioni per la griglia

È possibile usare la proprietà *Options* della griglia in fase di progettazione per controllare il comportamento di base e l’aspetto della griglia in fase di esecuzione.

Quando un componente griglia viene collocato per la prima volta su una scheda in fase di progettazione, la proprietà *Options* nell'Object Inspector viene visualizzata con un segno + (più) per indicare che la proprietà *Options* può essere espansa per visualizzare una serie di proprietà booleane che è possibile impostare singolarmente. Per visualizzare e impostare queste proprietà, fare clic sul simbolo +. Sotto la proprietà *Options*, nell'Object Inspector, appare la lista delle opzioni. Il simbolo + si trasforma nel simbolo - (meno), che consente di comprimere la lista facendo clic su di esso.

La tabella seguente elenca le proprietà *Options* che è possibile impostare, e descrive il modo in cui influiscono sulla griglia in fase di esecuzione.

Tabella 19.5 Proprietà Options espansa di TBDGrid

Opzione	Funzione
dgEditing	**true**: (Impostazione predefinita). Consente la modifica, l'inserimento e la cancellazione dei record della griglia. **false**: Disabilita la modifica, l'inserimento e la cancellazione dei record della griglia.
dgAlwaysShowEditor	**true**: Quando un campo viene selezionato, è nello stato Edit. **false**: (Impostazione predefinita). Un campo non è automaticamente nello stato Edit quando viene selezionato.
dgTitles	**true**: (Impostazione predefinita). Visualizza i nomi dei campi in cima alla griglia. **false**: Disattiva la visualizzazione dei nomi dei campi.
dgIndicator	**true**: (Impostazione predefinita). L'indicatore di colonna appare a sinistra della griglia, e l'indicatore del record attivo (una freccia a sinistra della griglia) è attivato per mostrare il record attivo. In fase di inserimento, la freccia diventa un asterisco. In fase di modifica, diventa un I-beam. **false**: L'indicatore di colonna è disattivato.
dgColumnResize	**true**: (Impostazione predefinita). Le colonne possono essere ridimensionate trascinando i relativi righelli nell'area del titolo. Il ridimensionamento modifica la larghezza corrispondente del componente TField sottostante. **false**: Le colonne non possono essere ridimensionate nella griglia.
dgColLines	**true**: (Impostazione predefinita). Visualizza delle righe di divisione verticale tra le colonne. **false**: Non visualizza delle righe di divisione verticale tra le colonne.
dgRowLines	**true**: (Impostazione predefinita). Visualizza delle righe di divisione verticale tra i record. **false**: Non visualizza delle righe di divisione verticale tra i record.
dgTabs	**true**: (Impostazione predefinita). Attiva lo spostamento tra i campi dei record. **false**: La pressione del tasto di tabulazione fa uscire dal controllo griglia.
dgRowSelect	**true**: La barra di selezione si espande all'intera larghezza della griglia. **false**: (Impostazione predefinita). La selezione di un campo in un record seleziona solo quel campo.

Tabella 19.5 Proprietà Options espansa di TBDGrid (continua)

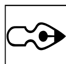
Opzione	Funzione
dgAlwaysShowSelection	**true**: (Impostazione predefinita). La barra di selezione nella griglia è sempre visibile, anche se il fuoco è su un altro controllo. **false**: La barra di selezione nella griglia è visibile solo quando il fuoco è sulla griglia.
dgConfirmDelete	**true**: ((Impostazione predefinita). Chiede conferma per la cancellazione dei record (*Ctrl+Canc*)). **false**: Cancella i record senza chiedere alcuna conferma.
dgCancelOnExit	**true**: (Default). Impostazione predefinita). Annulla un inserimento in sospeso quando il fuoco si sposta dalla griglia. Questa opzione impedisce l'invio accidentale di record parziali o vuoti. **false**: Consente gli inserimenti in sospeso.
dgMultiSelect	**true**: Consente all'utente di selezionare righe non contigue nella griglia usando i tasti *Ctrl+Maiusc* o *Maiusc+ freccia*. **false**: (Impostazione predefinita). Non consente all'utente di selezionare più righe.

Editing nella griglia

In fase di esecuzione si può usare una griglia per la modifica di dati esistenti o per l'immissione di nuovi record, se vengono soddisfatte le seguenti condizioni predefinite:

- La proprietà *CanModify* del Dataset è **true**.
- La proprietà *ReadOnly* della griglia è **false**.

Quando un utente modifica un record nella griglia, le modifiche apportate a ciascun campo vengono inviate in un buffer interno per i record, ma non vengono inoltrate finché l'utente non si sposta in un record diverso della griglia. Anche se il fuoco si sposta su un altro controllo di una scheda, la griglia non inoltra le modifiche finché il cursore del dataset non viene spostato su un altro record. Quando un record viene inoltrato, il dataset controlla tutti i componenti associati ai dati collegati per verificare se ci sono modifiche di stato. Se sorge un problema nell'aggiornamento dei campi contenenti i dati modificati, la griglia solleva un'eccezione e non modifica il record.



Se l'applicazione conserva gli aggiornamenti in cache, la conferma delle modifiche a un record si limita ad aggiunge tali modifiche a una cache interna. Esse non vengono registrare nella tabella del database sottostante finché l'applicazione non applica gli aggiornamenti.

Premendo il tasto *Esc* in qualunque campo, si possono annullare tutte le modifiche apportate a un record prima di passare su un altro.

Controllo del disegno della griglia

Il primo livello di controllo su come un controllo griglia viene disegnata è l'impostazione delle proprietà delle colonne. La griglia usa automaticamente le

proprietà del font, del colore e di allineamento di una colonna per tracciare le relative celle. Il testo dei campi dati viene riportato usando le proprietà *DisplayFormat* o *EditFormat* del componente field associato alla colonna.

È possibile potenziare tramite codice la logica predefinita di visualizzazione della griglia in un evento *OnDrawColumnCell* della griglia. Se la proprietà *DefaultDrawing* della griglia è **true**, il normale disegno viene eseguito prima che venga chiamato il gestore di evento *OnDrawColumnCell*. Il codice può poi intervenire sopra il piano di visualizzazione predefinito. Ciò è particolarmente utile quando è stata definita una colonna persistente vuota e si vogliono tracciare particolari elementi grafici nelle celle di tale colonna.

Se si intende sostituire completamente la logica di disegno della griglia, occorre impostare *DefaultDrawing* a **false** e collocare il codice di disegno nell'evento *OnDrawColumnCell* della griglia. Se si preferisce sostituire la logica di disegno solo in determinate colonne o in certi tipi di campi dati, si può chiamare *DefaultDrawColumnCell* all'interno del gestore di evento *OnDrawColumnCell* per far sì che la griglia usi il codice normale di disegno per le colonne selezionate. Ciò riduce, per esempio, la quantità di lavoro necessaria per cambiare solo il modo in cui vengono tracciati i tipi di campo Boolean.

Risposta alle azioni dell'utente in fase di esecuzione

È possibile modificare il comportamento della griglia scrivendo dei gestori di evento che rispondano ad azioni specifiche all'interno della griglia in fase di esecuzione.Dal momento che una griglia generalmente visualizza contemporaneamente molti campi e record, è possibile che si abbiano esigenze molto specifiche per rispondere ai cambiamenti nelle singole colonne. Per esempio, si potrebbe decidere di attivare e disattivare un pulsante in un punto qualunque della scheda ogni volta che un utente entra ed esce da una determinata colonna.

La tabella seguente elenca gli eventi di una griglia disponibili nell'Object Inspector.

Tabella 19.6 Eventi di controllo della griglia

Evento	Funzione
OnCellClick	Si verifica quando un utente fa clic su una cella nella griglia.
OnColEnter	Si verifica quando un utente si sposta in una colonna nella griglia.
OnColExit	Si verifica quando un utente lascia una colonna nella griglia.
OnColumnMoved	Si verifica quando l'utente sposta una colonna in una nuova posizione.
OnDblClick	Si verifica quando un utente fa doppio clic nella griglia.
OnDragDrop	Si verifica quando un utente effettua un'operazione 'drag-and-drop' nella griglia.
OnDragOver	Si verifica quando un utente trascina sopra la griglia.
OnDrawColumnCell	Si verifica quando l'applicazione ha bisogno di disegnare singole celle.
OnDrawDataCell	(obsoleto) Si verifica quando l'applicazione ha bisogno di disegnare singole celle se *State* è *csDefault*.
OnEditButtonClick	Si verifica quando l'utente fa clic su un pulsante ellissi in una colonna.
OnEndDrag	Si verifica quando un utente interrompe il trascinamento nella griglia.

Tabella 19.6 Eventi di controllo della griglia (continua)

Evento	Funzione
OnEnter	Si verifica quando la griglia ottiene il fuoco.
OnExit	Si verifica quando la griglia perde il fuoco.
OnKeyDown	Si verifica quando un utente preme un tasto o una combinazione di tasti sulla tastiera mentre si trova nella griglia.
OnKeyPress	Si verifica quando un utente preme un singolo tasto alfanumerico sulla tastiera mentre si trova nella griglia.
OnKeyUp	Si verifica quando un utente rilascia un tasto mentre si trova nella griglia.
OnStartDrag	Si verifica quando un utente inizia il trascinamento nella griglia.
OnTitleClick	Si verifica quando un utente fa clic sul titolo di una colonna.

Ci sono molti usi per questi eventi. Per esempio, si potrebbe scrivere un gestore per l'evento *OnDblClick* che richiami un elenco da cui un utente possa scegliere un valore da inserire in una colonna. Tale gestore userebbe la proprietà *SelectedField* per determinare la riga e la colonna attive.

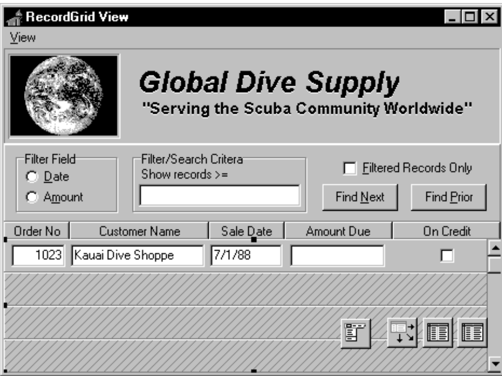
Creazione di una griglia contenente altri controlli associati ai dati

Un controllo *TDBCtrlGrid* visualizza più campi in più record in un formato griglia tabellare. Ogni cella della griglia mostra più record di un'unica riga.Per usare un controllo griglia per database:

1. Collocare un controllo griglia per database su una scheda.
2. Impostare la proprietà *DataSource* della griglia al nome di un datasource.
3. Collocare i singoli controlli dati all'interno della cella di progettazione per la griglia. La cella di progettazione è quella più in alto o più a sinistra nella griglia, ed è l'unica cella in cui si possono collocare altri controlli.
4. Impostare la proprietà *DataField* per ciascun controllo dati al nome di un campo. Il datasource per questi controlli è già impostato a quello del controllo griglia per database.
5. Sistemare i controlli all'interno della cella nel modo desiderato.

Quando si compila e si esegue un'applicazione che contiene un controllo griglia per database, la sistemazione dei controlli dati impostata nella cella di progettazione in progettazione viene riprodotta in tutte le celle della griglia. Ogni cella visualizza un record diverso di un dataset.

Figura 19.5 TDBCtrlGrid in fase di progettazione



La tabella seguente riporta alcune delle proprietà specifiche delle griglie controllo database che è possibile impostare in progettazione:

Tabella 19.7 Proprietà del controllo griglia per database selezionato

Proprietà	Funzione
AllowDelete	**true** (impostazione predefinita): Permette la cancellazione dei record.
	false: Impedisce la cancellazione dei record.
AllowInsert	**true** (default): impostazione predefinita): Permette l'inserimento dei record.
	false: Impedisce l'inserimento dei record.
ColCount	Imposta il numero delle colonne della griglia. Impostazione predefinita = 1.
Orientation	*goVertical* (impostazione predefinita): Visualizza i record dall'alto verso il basso.
	goHorizontal: Visualizza i record da sinistra a destra.
PanelHeight	Imposta l'altezza di un singolo pannello. Impostazione predefinita = 72.
PanelWidth	Imposta la larghezza di un singolo pannello. Impostazione predefinita = 200.
RowCount	Imposta il numero di pannelli da visualizzare. Impostazione predefinita = 3.
ShowFocus	**true** (impostazione predefinita): Visualizza un rettangolo relativo al fuoco intorno al pannello del record attivo in esecuzione.
	false: Non visualizza un rettangolo relativo al fuoco.

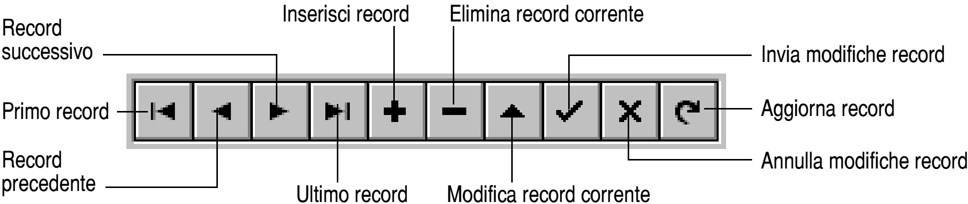
Per ulteriori informazioni sulle proprietà e sui metodi delle griglie controllo per database, consultare la online *VCL Reference*.

Spostamento e gestione dei record

TDBNavigator mette a disposizione degli utenti un semplice controllo per spostarsi attraverso i record di un dataset e per gestirli. Il 'navigatore' è costituito da una serie di pulsanti che consentono a un utente di spostarsi in avanti o a ritroso attraverso i record di un record alla volta, di andare al primo record o di passare all'ultimo, di inserire un nuovo record o di aggiornarne uno esistente, di inserire le modifiche ai dati o di annullarle, di cancellare un record o di aggiornarne la visualizzazione.

La Figura 19.6 mostra il navigatore che appare per impostazione predefinita quando, in fase di progetto, lo si colloca su una scheda. Il navigatore è composto da una serie di pulsanti che permettono a un utente di spostarsi da un record a un altro di un dataset e di modificare, cancellare, inserire e assegnare record. La proprietà *VisibleButtons* del navigatore consente di nascondere o di mostrare dinamicamente un sottoinsieme di questi pulsanti

Figura 19.6 Pulsanti del controllo TDBNavigator



La tabella che segue descrive i pulsanti del navigatore.

Tabella 19.8 Pulsanti di TDBNavigator

Pulsante	Funzione
First	Chiama il metodo *First* del dataset per impostare il record corrente al primo record.
Prior	Chiama il metodo *Prior* del dataset per impostare il record corrente al record precedente.
Next	Chiama il metodo *Next* del dataset per impostare il record corrente al record successivo.
Last	Chiama il metodo *Last* del dataset per impostare il record corrente all'ultimo record.
Insert	Chiama il metodo *Insert* del dataset per inserire un nuovo record prima del record corrente e impostare il dataset nello stato Insert.
Delete	Cancella il record attuale. Se la proprietà *ConfirmDelete* è true prima di cancellare chiede la conferma.
Edit	Pone il dataset nello stato Edit in modo da consentire la modifica del record attuale.
Post	Scrive le modifiche apportate al record corrente nel database.
Cancel	Cancella le modifiche al record corrente e riporta il dataset allo stato Browse.
Refresh	Azzera i buffer di visualizzazione dei controlli dati, quindi aggiorna i propri buffer dalla tabella fisica o dalla query. È utile nel caso in cui i dati associati siano stati stati modificati da un'altra applicazione.

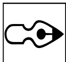
Scelta dei pulsanti del navigatore da visualizzare

Quando, in fase di progetto, si colloca per la prima volta *TDBNavigator* su una scheda, tutti i relativi pulsanti sono visibili. Per disattivare i pulsanti che non si vogliono usare su una scheda, si può usare la proprietà *VisibleButtons*. Ad esempio, quando si opera con un dataset unidirezionale, solo i pulsanti *First*, *Next* e *Refresh* sono significativi. Su una scheda nata per operazioni di consultazione anziché di modifica, ad esempio, si potrebbero disattivare i pulsanti *Edit*, *Insert*, *Delete*, *Post* e *Cancel*.

Visualizzazione e occultamento dei pulsanti del navigatore in fase di progettazione

La proprietà *VisibleButtons* nell'Object Inspector viene visualizzata con un segno + per indicare che può essere espansa per visualizzare un valore booleano per ciascun pulsante sul navigatore. Per vedere e impostare questi valori, fare clic sul segno +. L'elenco dei pulsanti che possono essere attivati o disattivati appare nell'Object Inspector sotto la proprietà *VisibleButtons*. Il simbolo + (più) si trasforma nel simbolo - (meno), che consente di comprimere la lista delle proprietà facendo clic su di esso.

La visibilità del pulsante è indicata dallo stato *Boolean* del valore del pulsante. Se il valore è impostato a true, il pulsante appare nel *TDBNavigator*. Se è **false**, il pulsante viene rimosso dal navigatore in fase di progetto e durante l'esecuzione.



Non appena i valori del pulsante sono impostati a **false**, vengono rimossi da *TDBNavigator* sulla scheda e i pulsanti che rimangono vengono espansi in ampiezza per riempire il controllo. È possibile trascinare le maniglie del controllo per ridimensionare i pulsanti.

Visualizzazione e occultamento dei pulsanti del navigatore in fase di esecuzione

In fase di esecuzione è possibile nascondere o mostrare i pulsanti del navigatore in risposta ad azioni dell'utente o a stati dell'applicazione. Si supponga, ad esempio, di fornire un singolo navigatore per spostarsi tra due dataset diversi, uno dei quali consente agli utenti di modificare record e l'altro è a sola lettura. Quando si passa da un dataset all'altro, l'utente potrebbe voler nascondere i pulsanti Insert, *Delete*, *Edit*, *Post*, *Cancel* e *Refresh* del dataset a sola lettura e mostrarli per l'altro dataset.

Si supponga di voler impedire la modifica di *OrdersTable* nascondendo i pulsanti *Insert*, *Delete*, *Edit*, *Post*, *Cancel* e *Refresh* del navigatore, ma di volere anche permettere la modifica di CustomersTable. La proprietà *VisibleButtons* controlla quali pulsanti vengono visualizzati nel navigatore. Di seguito è riportato un possibile modo per scrivere il codice del gestore di evento:

```
void __fastcall TForm1::CustomerCompanyEnter(TObject *Sender)
{
  if (Sender == (TObject *)CustomerCompany)
  {
    DBNavigatorAll->DataSource = CustomerCompany->DataSource;
    DBNavigatorAll->VisibleButtons = TButtonSet() << nbFirst << nbPrior << nbNext << nbLast;
  }
  else
  {
    DBNavigatorAll->DataSource = OrderNum->DataSource;
    DBNavigatorAll->VisibleButtons = TButtonSet() << nbInsert << nbDelete << nbEdit
                                << nbPost << nbCancel << nbRefresh;
  }
}
```

Visualizzazione del fumetto guida

Per visualizzare in esecuzione il fumetto guida per ciascun pulsante del navigatore, impostare la proprietà *ShowHint* del navigatore a **true**. Quando *ShowHint* è **true**, il navigatore visualizza una miniguida di suggerimento ogni volta che si passa il cursore del mouse sui pulsanti del navigatore. Per impostazione predefinita *ShowHint* è **false**.

La proprietà *Hints* controlla il testo del fumetto guida per ciascun pulsante. Per impostazione predefinita, *Hints* è una lista di stringhe vuota. Quando *Hints* è vuota, ogni pulsante del navigatore visualizza il testo guida predefinito. Per fornire un fumetto guida personalizzato per i pulsanti del navigatore, usare lo String editor per registrare nella proprietà *Hints* una riga separata di testo per il suggerimento associato a ciascun pulsante. Se presenti, le stringhe create sovrascrivono i suggerimenti predefiniti forniti dal controllo navigatore.

Uso di un singolo navigatore per più dataset

Come avviene per gli altri controlli data-aware, la proprietà *DataSource* di un navigatore specifica il datasource che collega il controllo a un dataset. Modificando la proprietà DataSource di un navigatore in esecuzione, un singolo navigatore può consentire lo spostamento tra i record e la gestione di più dataset.

Si supponga che una scheda contenga due controlli di modifica collegati rispettivamente ai dataset *CustomersTable* e *OrdersTable* tramite le sorgenti dati *CustomersSource* e *OrdersSource*. Quando un utente seleziona il controllo di modifica connesso a *CustomersSource*, anche il navigatore dovrebbe usare *CustomersSource* e quando l'utente seleziona il controllo di modifica connesso a *OrdersSource*, anche il navigatore dovrebbe passare a *OrdersSource*. È possibile programmare il gestore di evento *OnEnter* di uno dei controlli di modifica e quindi condividere quell'evento con l'altro controllo di modifica. Per esempio:

```
void __fastcall TForm1::CustomerCompanyEnter(TObject *Sender)
{
  if (Sender == (TObject *)CustomerCompany)
    DBNavigatorAll->DataSource = CustomerCompany->DataSource;
  else
    DBNavigatorAll->DataSource = OrderNum->DataSource;
}
```

Capitolo

20

Uso dei componenti di supporto decisionale

I componenti di supporto decisionale aiutano a creare tabelle incrociate - dette anche crosstab - e grafici. È possibile quindi usare queste tabelle e grafici per vedere e riassumere i dati da prospettive diverse. Per maggiori informazioni sui dati incrociati, consultare "I crosstab" a pagina 20-2.

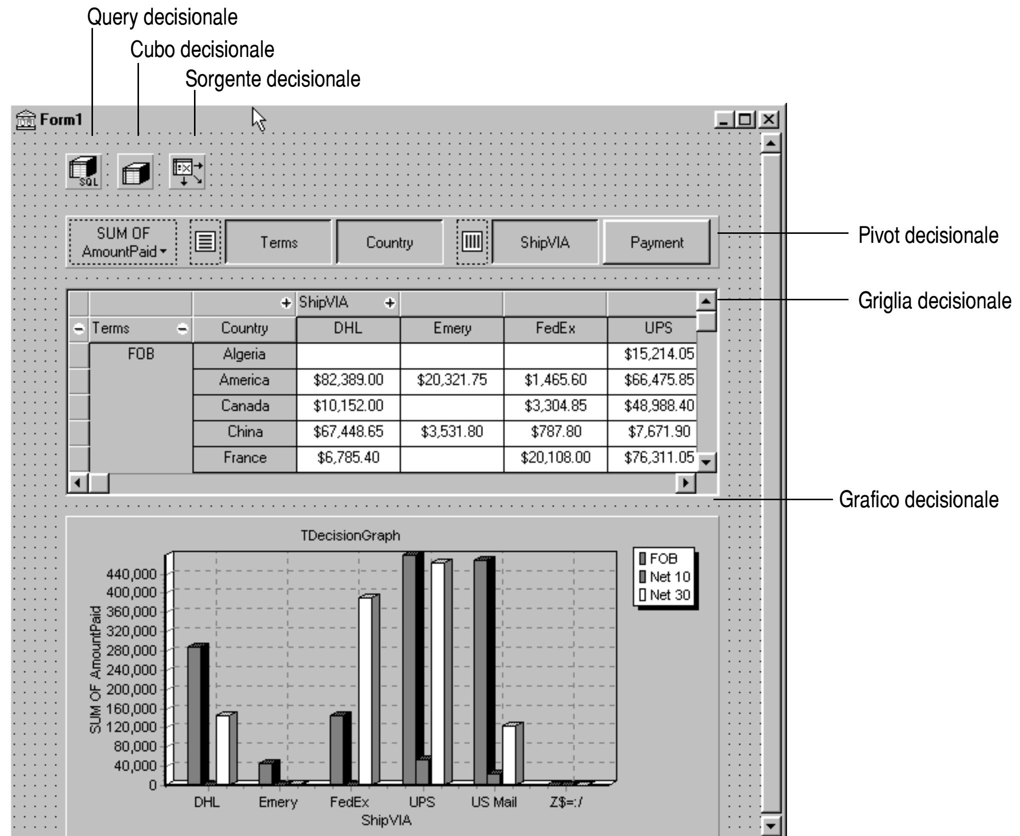
Panoramica

I componenti di supporto decisionale appaiono sulla pagina Decision Cube della tavolozza Component:

- Il cubo decisionale, *TDecisionCube*, è un magazzino dati multidimensionale.
- La sorgente decisionale, *TDecisionSource*, definisce lo stato del pivot attivo di una griglia o di un grafico decisionale.
- La query decisionale, *TDecisionQuery*, è una forma specializzata di *TQuery* usata per definire i dati in un cubo decisionale.
- Il pivot decisionale, *TDecisionPivot*, consente di ingrandire o di rimpicciolire le dimensioni del cubo decisionale o i campi con la semplice pressione di pulsanti.
- La griglia decisionale, *TDecisionGrid*, mostra dati mono e multidimensionali in forma tabellare.
- Il grafico decisionale, *TDecisionGraph*, mostra i campi di una griglia decisionale come un grafico dinamico che cambia al variare delle dimensioni dei dati.

La Figura 20.1 mostra tutti i componenti di supporto decisionale collocati su una scheda in fase di progettazione.

Figura 20.1 Componenti di supporto decisionale in progettazione



I crosstab

Le tabelle incrociate, o crosstab, sono un modo di presentare sottoinsiemi di dati per rendere più evidenti relazioni e andamenti. I campi della tabella diventano le dimensioni del crosstab mentre i valori dei campi definiscono le categorie e i riepiloghi all'interno di una dimensione.

È possibile usare i componenti di supporto decisionale per disporre sulle schede i crosstab. *TDecisionGrid* mostra i dati in una tabella, mentre *TDecisionGraph* la traccia graficamente. *TDecisionPivot* dispone di pulsanti che semplificano la visualizzazione o l'occultamento delle dimensioni e il loro spostamento tra le colonne e le righe.

I crosstab possono essere monodimensionali o multidimensionali.

Crosstab monodimensionali

I crosstab monodimensionali mostrano una riga (o una colonna) di riepilogo per le categorie di una singola dimensione. Ad esempio, se la dimensione della colonna scelta è Payment e la categoria di riepilogo è Amount Paid, il crosstab in Figura 20.2 mostra l’importo pagato con ciascuno dei metodi.

Figura 20.2 Un crosstab mono-dimensionale

SUM OF AmountPaid | Terms | Country | ShipVIA | Payment

Payment					
AmEx	Cash	Check	COD	Credit	MC
$134,753.40	$164,003.65	$270,492.15	$33,776.55	$1,332,430.25	$250,163.25

Crosstab multidimensionali

I crosstab multidimensionali usano dimensioni supplementari per le righe e/o per le colonne. Per esempio, un crosstab bidimensionale può mostrare gli importi pagati in base al metodo di pagamento per ciascun paese.

Un crosstab tridimensionale può mostrare gli importi pagati in base al metodo e alle condizioni di pagamento paese per paese, come mostrato nella Figura 20.3.

Figura 20.3 Crosstab tridimensionale

SUM OF AmountPaid | Terms | Country | ShipVIA | Payment

Terms	Country	Check	COD	Credit	MC
FOB	Algeria	$2,577.85		$1,400.00	$13,814.05
	America			$356,816.20	$20,881.35
	Canada			$24,485.00	$3,304.85
	China	$61,936.90		$6,641.55	

Regole generali per l’uso dei componenti di supporto decisionale

I componenti di supporto decisionale, elencati a pagina 20-1,possono essere usati contemporaneamente per presentare dati multidimensionali sotto forma di tabelle e di grafici. Ad ogni dataset possono essere collegati più tabelle o grafici. In esecuzione, è possibile usare più istanze di *TDecisionPivot* per visualizzare dati di prospettive differenti.

Per creare una scheda con tabelle e grafici di dati multidimensionali, procedere nel modo seguente:

1 Creare una scheda.

2 Aggiungere questi componenti alla scheda (usando poi l'Object Inspector per collegarli come indicato di seguito):

- Un dataset, di solito *TDecisionQuery* (per maggiori informazioni, consultare "Creazione di dataset decisionali con l'editor Decision Query" a pagina 20-6) o *TQuery*
- Un cubo decisionale, *TDecisionCube*, collegato al dataset impostando la proprietà *DataSet* al nome del dataset
- Una sorgente decisionale, *TDecisionSource*, collegato al cubo decisionale impostando la proprietà di *DecisionCube* al nome del cubo decisionale

3 Aggiungere un pivot decisionale, *TDecisionPivot*, e collegarlo alla sorgente decisionale con l'Object Inspector impostando la proprietà *DecisionSource* al nome della sorgente decisionale appropriata. Il pivot decisionale è facoltativo ma utile; permette allo sviluppatore e agli utenti finali della scheda di modificare le dimensioni visualizzate in griglie o in grafici decisionali con la semplice pressione di pulsanti.

Nel loro orientamento predefinito, quello orizzontale, i pulsanti sul lato sinistro del pivot decisionale agiscono sui campi posti sul lato sinistro della griglia decisionale (le righe); i pulsanti sul lato destro agiscono sui campi della parte superiore della griglia decisionale (le colonne).

È possibile determinare dove fare apparire i pulsanti del pivot decisionale impostando la proprietà *GroupLayout* a *xtVertical*, *xtLeftTop* o *xtHorizontal* (impostazione predefinita). Per maggiori informazioni sulle proprietà del pivot decisionale, consultare "Uso dei pivot decisionali" a pagina 20-10.

4 Aggiungere una o più griglie e grafici decisionali, collegandoli alla sorgente decisionale. Per maggiori informazioni, consultare "Creazione e uso delle griglie decisionali" a pagina 20-11 e "Creazione e uso dei grafici decisionali" a pagina 20-13.

5 Usare l'editor della query decisionale o la proprietà *SQL* di *TDecisionQuery* (o di TQuery) per specificare le tabelle, i campi e i riepiloghi da visualizzare nella griglia o nel grafico. L'ultimo campo dell'istruzione SQL SELECT deve essere il campo di riepilogo. Gli altri campi nell'istruzione SELECT devono essere i campi di GROUP BY. Per istruzioni, consultare "Creazione di dataset decisionali con l'editor Decision Query" a pagina 20-6.

6 Impostare la proprietà *Active* della query decisionale (o del componente dataset alternativo) a **true**.

7 Usare la griglia e il grafico decisionali per mostrare e tracciare sotto forma di grafico dimensioni dati differenti. Per istruzioni e suggerimenti, consultare "Uso delle griglie decisionali" a pagina 20-11 e "Uso dei grafici decisionali" a pagina 20-14.

Per vedere un'illustrazione che riporti in una scheda tutti i componenti di supporto decisionale, consultare Figura 20.1 a pagina 20-2.

Uso dei dataset con i componenti di supporto decisionale

L'unico componente di supporto decisionale collegato direttamente a un dataset è il cubo decisionale, *TDecisionCube*. *TDecisionCube* si aspetta ricevere i dati tramite i gruppi e i riepiloghi definiti da un'istruzione SQL di un formato accettabile. La frase GROUP BY deve contenere gli stessi campi non riepilogati (e nello stesso ordine) contenuti nella frase SELECT, e i campi di riepilogo devono essere identificati.

Il componente query decisionale, *TDecisionQuery*, è una versione specializzata di *TQuery*. È possibile usare TDecisionQuery per definire più semplicemente la configurazione delle dimensioni (righe e colonne) e i valori di riepilogo usati per fornire i dati ai cubi decisionali, *TDecisionCube*. Come dataset per *TDecisionCube* un è possibile anche usare un comune *TQuery* o altri dataset abilitati a BDE, ma la corretta configurazione del dataset e di *TDecisionCube* sono di competenza del progettista.

Per operare correttamente col cubo decisionale, tutti i campi progettati nel dataset devono essere dimensioni o riepiloghi. I riepiloghi devono essere valori aggiuntivi (come somma o conteggio) e devono rappresentare i totali per ciascuna combinazione dei valori di dimensione. Per semplificare al massimo la configurazione, nel dataset le somme devono essere chiamate "Sum..." mentre i conteggi dovrebbero essere chiamati "Count...".

Decision Cube può calcolare correttamente pivot, totali parziali e in dettaglio solo per i riepiloghi le cui celle sono di tipo additivo. (SUM e COUNT sono celle additive, AVERAGE, MAX e MIN no). La crosstab di costruzione del pivot visualizza solo le griglie che contengono totali additivi. Se si usano totali non additivi, ricorrere ad una griglia decisionale statica che non sia pivot, dettaglio o totale parziale.

Poiché le medie possono essere calcolate usando SUM diviso per COUNT, viene aggiunta automaticamente una media pivot quando le dimensioni SUM e COUNT per un campo vengono incluse nel dataset. Questo tipo di media deve essere utilizzato preferibilmente per una media calcolata usando un'istruzione AVERAGE.

Le medie possono essere calcolate anche usando COUNT(*). A questo scopo, includere un selettore "COUNT(*) COUNTALL" nella query. Se per calcolare le medie si usa COUNT(*), il singolo aggregatore può essere usato per tutti i campi. COUNT(*) deve essere utilizzato solo nei casi in cui nessuno dei campi che deve essere riepilogato include valori vuoti, o in cui non è disponibile un totale COUNT per ogni campo.

Creazione di dataset decisionali con TQuery o TTable

Se si usa come dataset decisionale un comune componente *TQuery*, si deve preparare manualmente l'istruzione SQL, facendo attenzione a fornire una frase GROUP BY che contenga gli stessi campi (e nello stesso ordine) della frase SELECT.

L'istruzione SQL può essere simile a questa:

```
SELECT ORDERS."Terms", ORDERS."ShipVIA",
  ORDERS."PaymentMethod", SUM( ORDERS."AmountPaid" )
FROM "ORDERS.DB" ORDERS
GROUP BY ORDERS."Terms", ORDERS."ShipVIA", ORDERS."PaymentMethod"
```

L'ordinamento dei campi SELECT deve coincidere con quello di GROUP BY.

Con *TTable* occorre dire al cubo decisionale quali sono i campi della query da considerare campi di raggruppamento e quali quelli da considerare campi di riepilogo. Per eseguire questa operazione, compilare Dimension Type per ciascun campo in *DimensionMap* della pagina Decision Cube. Occorre indicare se ciascun campo è una dimensione o un riepilogo, e se un riepilogo, si deve fornirne il tipo. Poiché le medie pivot dipendono dai calcoli SUM/COUNT, occorre fornire anche il nome base del campo per consentire al cubo decisionale di confrontare le coppie di aggregatori SUM e COUNT.

Creazione di dataset decisionali con l'editor Decision Query

Tutti i dati usati dai componenti di supporto decisionale passano attraverso il cubo decisionale che accetta un dataset formattato in modo particolare, prodotto molto facilmente da una query SQL. Per maggiori informazioni, consultare "Uso dei dataset con i componenti di supporto decisionale" a pagina 20-5.

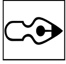
Anche se è possibile usare *TTable* e *TQuery* come dataset decisionali, risulta più semplice ricorrere a *TDecisionQuery*, l'editor della query decisionale fornito insieme al prodotto. Può essere usato per specificare tabelle, campi e riepiloghi che devono apparire nel cubo decisionale contribuendo a configurare correttamente le porzioni SELECT e GROUP BY della query SQL.

Per usare l'editor Decision Query:

1. Selezionare il componente della query decisionale sulla scheda, poi fare clic col tasto destro e scegliere Decision Query editor. Apparirà la finestra di dialogo Decision Query editor.
2. Selezionare il database da utilizzare.
3. Per query su una singola tabella, fare clic sul pulsante Select Table.

   Per query più complesse, che coinvolgono join su più tabelle, fare clic sul pulsante Query Builder per visualizzare SQL Builder oppure, nella pagina SQL, scrivere l'istruzione SQL utilizzando la casella di testo apposita.
4. Ritornare alla finestra di dialogo Decision Query editor.
5. Nella finestra di dialogo Decision Query editor selezionare i campi nell'elenco Available Fields e assegnarli a Dimensions o a Summaries facendo clic sul pulsante freccia a destra opportuno. Mano mano che si aggiungono i campi all'elenco Summaries, selezionare dal menu visualizzato il tipo di riepilogo da utilizzare: somma, conteggio o media.
6. Per impostazione predefinita, tutti i campi e i riepiloghi definiti nella proprietà *SQL* della query decisionale appaiono nelle caselle di riepilogo Active Dimensions e Active Summaries. Per eliminare una dimensione o riepilogo, selezionarlo nell'elenco e fare clic sulla freccia sinistra accanto all'elenco, o fare doppio clic sull'elemento da eliminare. Per aggiungerlo all'elenco originale, selezionarlo nella casella di riepilogo Available Fields e fare clic sulla freccia destra opportuna.

Una volta definiti i contenuti del cubo decisionale, è possibile modificare ulteriormente le dimensioni di visualizzazione agendo sulla proprietà *DimensionMap* e sui pulsanti di *TDecisionPivot*. Per maggiori informazioni, consultare le sezioni successive "Uso dei cubi decisionali", "Uso delle sorgenti decisionali" a pagina 20-9, e "Uso dei pivot decisionali" a pagina 20-10.



Quando si usa l'editor Decision Query, la query inizialmente viene gestita nella sintassi SQL ANSI-92, quindi tradotta (se necessario) nel dialetto usato dal server. L'editor Decision Query legge e visualizza solo il formato SQL ANSI standard. La traduzione del dialetto viene assegnata automaticamente alla proprietà SQL di *TDecisionQuery*. Per modificare una query, effettuare l'editing della versione ANSI-92 in Decision Query anziché nella proprietà SQL.

Uso dei cubi decisionali

Il componente cubo decisionale, *TDecisionCube*, è un'area di memorizzazione dati multidimensionale che recupera i dati da un dataset (di solito usando un'istruzione SQL, strutturata in modo particolare, immessa con *TDecisionQuery* o *TQuery*). I dati sono memorizzati in un formato che ne semplifica l'utilizzo come pivot (ovvero, modifica il modo in cui i dati vengono organizzati e riepilogati) senza dover eseguire la query una seconda volta.

Proprietà del cubo decisionale e eventi

Le proprietà *DimensionMap* di *TDecisionCube* non solo controllano quali dimensioni e riepiloghi far apparire, ma consentono anche di impostare gli intervalli di dati e di specificare il numero massimo di dimensioni che il cubo decisionale può supportare. Si può anche indicare se visualizzare o meno i dati in fase di progetto. È possibile visualizzare nomi (le categorie), valori, totali parziali o dati. La visualizzazione dei dati in fase di progetto può essere dispendiosa in termini di tempo, a seconda della sorgente dati utilizzata.

Facendo clic sul pulsante con le ellissi accanto a *DimensionMap* nell'Object Inspector, appare la finestra di dialogo Decision Cube editor. È possibile usarne le pagine e i controlli per impostare le proprietà di *DimensionMap*.

L'evento *OnRefresh* viene attivato ogni qualvolta si rigenera la cache del cubo decisionale. Gli sviluppatori possono accedere alla nuova mappa della dimensione e modificarla per liberare memoria, modificare il valore massimo dei riepiloghi o delle dimensioni, e così via. *OnRefresh* è anche utile se gli utenti accedono all'editor del cubo decisionale; in questa fase il codice dell'applicazione può rispondere alle modifiche apportate dall'utente.

Uso di Decision Cube editor

È possibile usare l'editor del cubo decisionale per impostare le proprietà di *DimensionMap* dei cubi decisionali. L'editor del cubo decisionale può essere visualizzato tramite l'Object Inspector, come descritto nella sezione precedente, o

facendo clic, in fase di progetto, con il tasto destro su un cubo decisionale in una scheda e selezionando Decision Cube editor.

La finestra di dialogo Decision Cube Editor ha due separatori:

- Dimension Settings, usato per attivare o disattivare le dimensioni disponibili, cambiare il nome e riformattare le dimensioni, mettere le dimensioni in un stato permanentemente espanso e impostare gli intervalli dati da visualizzare.
- Memory Control, usato per impostare il massimo numero di dimensioni e riepiloghi che possono essere contemporaneamente attivi, per visualizzare informazioni sull'uso della memoria, e per determinare i nomi e i dati che appaiono in fase di progetto.

Visualizzazione e modifica delle impostazioni delle dimensioni

Per vedere le impostazioni delle dimensioni, visualizzare l'editor Decision Cube e fare clic sul separatore Dimension Settings. Quindi, selezionare una dimensione o un riepilogo nell'elenco Available Fields. Le informazioni appaiono nelle caselle sul lato destro dell'editor:

- Per modificare la dimensione o il nome del riepilogo che appaiono nel pivot, nella griglia o nel grafico decisionali, immettere un nuovo nome nella casella di testo Display Name.
- Per determinare se il campo selezionato è una dimensione o un riepilogo, leggere il testo nella casella di testo Type.Se il dataset è un componente *TTable*, è possibile usare Type per specificare se il campo selezionato è una dimensione o un riepilogo.
- Per disattivare o attivare la dimensione o il riepilogo selezionati, modificare l'impostazione nella casella di riepilogo a discesa Active Type: Active, As Needed o Inactive. La disattivazione di una dimensione o l'uso dell'impostazione As Needed fanno risparmiare memoria.
- Per modificare il formato di quella dimensione o quel riepilogo, immettere una stringa di formato nella casella di testo Format.
- Per visualizzare quella dimensione o quel riepilogo per Anno, Trimestre o Mese (Year, Quarter, o Month), modificare l'impostazione nella casella di riepilogo a discesa Binning. Per mettere la dimensione o il riepilogo selezionati in uno stato permanentemente compresso, si può scegliere l'opzione Set nella casella di riepilogo Binning. Ciò può essere utile per risparmiare memoria quando una dimensione ha molti valori. Per maggiori informazioni, consultaree "Componenti di supporto decisionale e controllo della memoria" a pagina 20-20.
- Per determinare i valori iniziali degli intervalli, o il valore drill-down per una dimensione "Set", scegliere innanzi tutto l'appropriato valore Grouping nell'elenco a discesa Grouping, quindi immettere il valore iniziale dell'intervallo o il valore drill-down permanente nell'elenco a discesa Initial Value.

Impostazione del numero massimo di dimensioni e di riepiloghi disponibili

Per determinare il numero massimo di dimensioni e di riepiloghi disponibili per i pivot, le griglie e i grafici decisionali associati al cubo decisionale selezionato,

visualizzare l'editor Decision Cube e fare clic sul separatore Memory Control. Usare i controlli di modifica per regolare, se necessario, le impostazioni attive. Queste aiutano a controllare la quantità di memoria necessaria per il cubo decisionale. Per maggiori informazioni, consultare "Componenti di supporto decisionale e controllo della memoria" a pagina 20-20.

Visualizzazione e modifica delle opzioni di progetto

Per determinare quante informazioni appaiono in fase di progetto, visualizzare il Decision Cube editor e fare clic sul separatore Memory Control. Quindi, controllare l'impostazione che indica quali nomi e dati visualizzare. La visualizzazione dei dati o dei nomi di campo in fase di progetto può causare a volte un degrado delle prestazioni dovuto al tempo necessario per recuperare i dati.

Uso delle sorgenti decisionali

Il componente sorgente decisionale, *TDecisionSource*, definisce lo stato attuale del pivot di griglie o di grafici decisionali. Qualunque coppia di oggetti che usa la stessa sorgente decisionale condivide anche gli stati del pivot.

Proprietà ed eventi

Di seguito sono elencate alcune proprietà ed eventi particolari che controllano l'aspetto e il comportamento delle sorgenti decisionali:

- La proprietà *ControlType* di *TDecisionSource* indica se i pulsanti del pivot decisionale devono agire come caselle di controllo (selezioni multiple) o come pulsanti di opzione (selezioni mutuamente esclusive).
- Le proprietà *SparseCols* e *SparseRows* di *TDecisionSource* indicano se visualizzare o meno le colonne o le righe senza valori; se **true**, vengono visualizzate le colonne o le righe rade.
- *TDecisionSource* ha i seguenti eventi:
  - *OnLayoutChange* si verifica quando l'utente esegue rotazioni o compressioni che riorganizzano i dati.
  - *OnNewDimensions* si verifica quando i dati vengono completamente modificati, come, ad esempio, quando vengono alterati i campi di riepilogo o delle dimensioni.
  - *OnSummaryChange* si verifica quando viene modificato il riepilogo attuale.
  - *OnStateChange* si verifica quando si attiva o si disattiva Decision Cube.
  - *OnBeforePivot* si verifica quando le modifiche sono già state confermate ma l'interfaccia utente non è stata ancora aggiornata. Gli sviluppatori hanno l'opportunità di apportare modifiche, per esempio, alla capacità o allo stato di pivot, prima che gli utenti dell'applicazione vedano il risultato delle azioni precedenti.

- *OnAfterPivot* si attiva dopo una modifica nello stato di pivot. In quel preciso momento, gli sviluppatori possono intercettare informazioni.

Uso dei pivot decisionali

Il componente pivot decisionale, *TDecisionPivot*, consente di aprire o di chiudere le dimensioni del cubo decisionale o i campi premendo dei pulsanti. Quando si apre una riga o una colonna premendo uno dei pulsanti di *TDecisionPivot*, la dimensione corrispondente appare sul componente *TDecisionGrid* o *TDecisionGraph*. Quando si chiude una dimensione, i dati particolareggiati non appaiono; la vista viene ridotta mostrando solo i totali delle altre dimensioni. Una dimensione può essere anche in uno stato compresso, in cui appaiono solamente i riepiloghi per un determinato valore del campo dimensione.

È inoltre possibile usare il pivot decisionale per riorganizzare le dimensioni visualizzate nella griglia decisionale e nel grafico decisionale. È sufficiente trascinare un pulsante sull'area della riga o della colonna o riordinare i pulsanti all'interno della stessa area.

Per le illustrazioni dei pivot decisionali in progettazione consultare le Figure 20.1, 20.2, e 20.3.

Proprietà dei pivot decisionali

Di seguito sono elencate alcune proprietà speciali che controllano l'aspetto e il comportamento dei pivot decisionali:

- Le prime proprietà elencate per *TDecisionPivot* ne definiscono il comportamento complessivo e l'aspetto. È possibile impostare *ButtonAutoSize* a **false** per fare in modo che *TDecisionPivot* impedisca l'espansione e la contrazione dei pulsanti mentre si sta modificando la dimensione del componente.
- La proprietà *Groups* di *TDecisionPivot* definisce quali pulsanti di dimensione dovranno apparire. È possibile visualizzare la riga, la colonna e i gruppi di pulsanti di selezione di riepilogo in qualsiasi combinazione. Si deve tenere presente che se si vuole maggior flessibilità sul posizionamento di questi gruppi, è possibile collocare in una posizione della scheda un componente *TDecisionPivot* che contiene solamente righe, e, in un'altra posizione, un secondo componente che contiene solamente colonne.
- Solitamente, *TDecisionPivo* viene aggiunto sopra a *TDecisionGrid*. In base all'orientamento predefinito, quello orizzontale, i pulsanti sul lato sinistro di *TDecisionPivot* agiscono sui campi posti sul lato sinistro di *TDecisionGrid* (le righe); i pulsanti sul lato destro agiscono sui campi posti nella parte superiore di *TDecisionGrid* (le colonne).
- È possibile determinare il punto in cui appaiono i pulsanti di *TDecisionPivot* impostandone la proprietà *GroupLayout* a *xtVertical*, *xtLeftTop*, o *xtHorizontal* (che è il valore predefinito ed è descritto nel paragrafo precedente).

Creazione e uso delle griglie decisionali

I componenti della griglia decisionale, *TDecisionGrid*, presentano i dati incrociati sotto forma di tabella. Queste tabelle, descritte a pagina 20-2, vengono anche chiamate crosstab. La Figura 20.1 a pagina 20-2 mostra una griglia decisionale su una scheda in progettazione.

Creazione di griglie decisionali

Per creare una scheda con uno o più tabelle di dati incrociati:

1 Seguire le istruzioni da 1 a 3 elencate nella sezione "Regole generali per l'uso dei componenti di supporto decisionale" a pagina 20-3.

2 Aggiungere uno o più componenti *TDecisionGrid* (griglia decisionale) e collegarli alla sorgente decisionale, *TDecisionSource*, con l'Object Inspector impostando la proprietà *DecisionSource* al componente sorgente decisionale opportuno.

3 Procedere con le istruzioni da 5 a 7 elencate nella sezione "Regole generali per l'uso dei componenti di supporto decisionale".

Per una descrizione di ciò che appare nella griglia decisionale e sul modo di utilizzarla, consultare la sezione "Uso delle griglie decisionali" a pagina 20-11.

Per aggiungere un grafico alla scheda, seguire le istruzioni riportate nella sezione "Creazione di grafici decisionali" a pagina 20-14.

Uso delle griglie decisionali

Il componente griglia decisionale, *TDecisionGrid*, visualizza i dati provenienti dai cubi decisionali (*TDecisionCube*) collegati a sorgenti decisionali (*TDecisionSource*).

Per impostazione predefinita, la griglia appare con i campi dimensione disposti sul lato sinistro e/o superiore, a seconda delle istruzioni di raggruppamento definite nel dataset. Le categorie, una per ciascun valore dei dati, appaiono sotto ciascun campo. È possibile

- Aprire e chiudere le dimensioni
- Riorganizzare o ruotare righe e colonne
- Espandere una riga per avere altri dettagli
- Limitare la selezione della dimensione a una singola dimensione per ciascun asse

Per maggiori informazioni sulle proprietà speciali e sugli eventi della griglia decisionale, consultare la sezione "Proprietà della griglia decisionale" a pagina 20-12.

Apertura e chiusura dei campi della griglia decisionale

Se in una dimensione o in un campo di riepilogo appare il segno più (+), significa che uno o più campi alla sua destra sono chiusi (nascosti). È possibile aprire campi e categorie supplementari facendo clic sul segno. Il segno meno (–) indica un campo

completamente aperto (espanso). Quando si fa clic sul segno, il campo si chiude. Questa opzione può essere disattivata; per maggiori informazioni, consultare il paragrafo "Proprietà della griglia decisionale" a pagina 20-12.

Riorganizzazione di righe e colonne in griglie decisionali

È possibile trascinare le intestazioni di riga e di colonna in nuove posizioni all'interno dello stesso asse o su un altro asse. In questo modo è possibile riorganizzare la griglia e vedere i dati da nuove prospettive man mano che cambia la modalità di raggruppamento. Questa opzione può essere disattivata; per maggiori informazioni, consultare il paragrafo "Proprietà della griglia decisionale" a pagina 20-12.

Se si include un pivot decisionale, è possibile premere e trascinare i pulsanti per riorganizzare la visualizzazione. Per istruzioni, consultare "Uso dei pivot decisionali" a pagina 20-10.

Espansione dei dettagli nelle griglie decisionali

È possibile espandere una dimensione per vedere più dettagli.

Ad esempio, se si fa clic col tasto destro su un'etichetta di categoria (intestazione di riga) per una dimensione mentre ce ne sono altre sotto di essa in forma contratta, è possibile decidere se espandere e consultare solamente i dati di quella categoria. Se una dimensione è espansa, sulla griglia non appaiono le etichette di categoria della dimensione, dato che vengono visualizzati solamente i record per un singolo valore della categoria. Se sulla scheda c'è un pivot decisionale, questo visualizza i valori di categoria e, volendo, consente di passare ad altri valori.

Per espandere una dimensione:

- Fare clic col tasto destro su un'etichetta di categoria e selezionare Drill In To This Value, oppure
- Fare clic col tasto destro su un pulsante pivot e scegliere Drilled In.

Per riattivare di nuovo la dimensione completa:

- Fare clic col tasto destro sul pulsante pivot corrispondente, oppure fare clic col tasto destro nell'angolo superiore-sinistro della griglia decisionale e selezionare la dimensione.

Limitazione della selezione delle dimensioni in griglie decisionali

Modificando la proprietà *ControlType* della sorgente decisionale si determina se è possibile selezionare più dimensioni per ciascun asse della griglia. Per maggiori informazioni, consultare "Uso delle sorgenti decisionali" a pagina 20-9.

Proprietà della griglia decisionale

Il componente griglia decisionale, *TDecisionGrid*, visualizza i dati provenienti dal componente *TDecisionCube* collegato a *TDecisionSource*. Per impostazione predefinita, i dati appaiono in una griglia con i campi di categoria disposti sul lato sinistro e nella parte superiore della griglia.

Di seguito sono elencate alcune proprietà speciali che controllano l'aspetto e il comportamento delle griglie decisionali:

- *TDecisionGrid* ha proprietà uniche per ciascuna dimensione.Per impostarle, scegliere *Dimensions* nell'Object Inspector, e selezionare poi una dimensione. A questo punto, le proprietà di quella dimensione appaiono nell'Object Inspector: *Alignment* definisce l'allineamento delle etichette di categoria per quella dimensione, *Caption* può essere usata per sostituire il nome predefinito della dimensione, *Color* definisce il colore delle etichette di categoria, *FieldName* visualizza il nome della dimensione attiva, *Format* può memorizzare qualsiasi formato standard per quel tipo di dati e *Subtotals* indica se visualizzare o meno i totali parziali per quella dimensione. Con i campi di riepilogo queste stesse proprietà vengono usate per modificare l'aspetto dei dati che appaiono nell'area di riepilogo della griglia. Una volta completata l'impostazione delle proprietà della dimensione, fare clic su un componente nella scheda o scegliere un componente nella casella di riepilogo a discesa nella parte superiore dell'Object Inspector.
- La proprietà *Options* di *TDecisionGrid* consente di controllare la visualizzazione delle linee della griglia (*cgGridLines* = **true**), permettendo l'attivazione delle opzioni secondarie (contrazione ed espansione delle dimensioni con indicatori + e - ; *cgOutliner* = **true**), e l'attivazione della rotazione tramite "drag and drop" (*cgPivotable* = **true**)).
- L'evento *OnDecisionDrawCell* di *TDecisionGrid* consente di modificare l'aspetto di ciascuna cella mentre viene disegnata. L'evento passa *String*, *Font* e *Color* della cella attiva come parametri di riferimento. È possibile modificare questi parametri per realizzare effetti come, ad esempio, colori speciali per indicare valori negativi. Oltre a *DrawState*, che è passato da *TCustomGrid*, l'evento passa *TDecisionDrawState*, che può essere usato per determinare quale tipo di cella viene disegnando. È possibile ottenere ulteriori informazioni sulla cella utilizzando le funzioni *Cells*, *CellValueArray*, o *CellDrawState*.
- L'evento *OnDecisionExamineCell* di *TDecisionGrid* consente di agganciare l'evento "clic destro" sulle celle dei dati, ed è stato creato per permettere al programma di visualizzare informazioni (come ad esempio i record di dettaglio) su quella determinata cella di dati. Quando l'utente fa clic col tasto destro su una cella di dati, all'evento vengono fornite tutte le informazioni usate per comporre i valori dei dati, compresi il valore di riepilogo attualmente attivo e un *ValueArray* di tutti i valori della dimensione usati per creare il valore di riepilogo.

Creazione e uso dei grafici decisionali

I componenti grafico decisionale, *TDecisionGraph*, presentano dati incrociati in formato grafico. Ciascun grafico decisionale mostra il valore di un singolo riepilogo, come Sum, Count, o Avg, tracciato per una o più dimensioni. Per maggiori informazioni sui crosstab, consultare la pagina 20-3. Per illustrazioni di grafici decisionali in fase di progetto, vedere la Figura 20.1 a pagina 20-2 e la Figura 20.4 a pagina 20-15.

Creazione di grafici decisionali

Per creare una scheda con uno o più grafici decisionali:

1. Seguire le istruzioni da 1 a 3 elencate nella sezione “Regole generali per l’uso dei componenti di supporto decisionale” a pagina 20-3.
2. Aggiungere uno o più componenti grafici decisionali (*TDecisionGraph*) e collegarli con l’Object Inspector alla sorgente decisionale, *TDecisionSource*, impostandone la proprietà *DecisionSource* al componente della sorgente decisionale opportuno.
3. Procedere con le istruzioni da 5 a 7 elencate nella sezione “Regole generali per l’uso dei componenti di supporto decisionale”.
4. Per finire, fare clic col tasto destro sul grafico e selezionare Edit Chart per modificare l’aspetto della serie del grafico. Per ciascuna dimensione del grafico è possibile impostare delle proprietà campione, e impostare poi singolarmente una serie di proprietà per sostituire quelle predefinite. Per maggiori informazioni, consultare la sezione “Personalizzazione dei grafici decisionali” a pagina 20-16.

Per una descrizione di ciò che appare nel grafico decisionale e del modo in cui usarlo, consultare la prossima sezione “Uso dei grafici decisionali”.

Per aggiungere una griglia decisionale o una tabella crosstab alla scheda, seguire le istruzioni della sezione “Creazione e uso delle griglie decisionali” a pagina 20-11.

Uso dei grafici decisionali

Il componente grafico decisionale, *TDecisionGraph*, visualizza i campi provenienti dalla sorgente decisionale (*TDecisionSource*) come un grafico dinamico che si modifica quando le dimensioni dei dati vengono aperte, chiuse, spostate con “drag-and-drop” o riordinate con il pivot decisionale (*TDecisionPivot*).

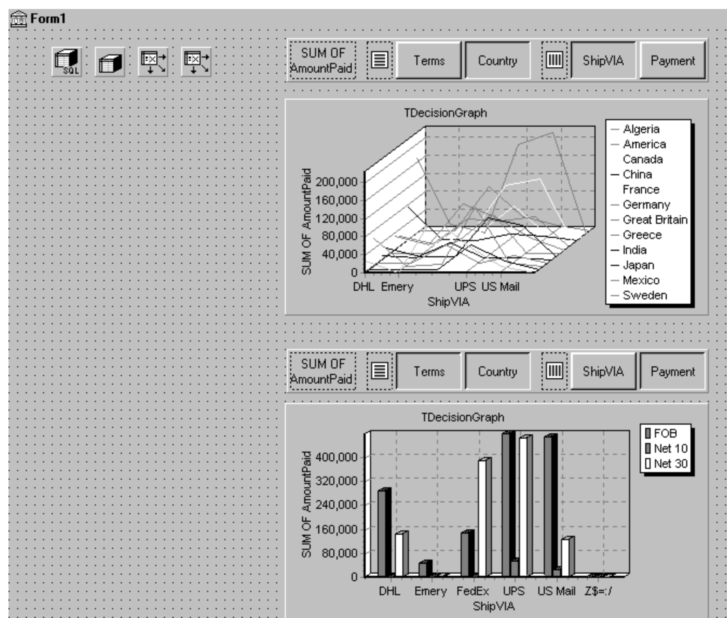
I dati trasformati in grafico provengono da un dataset specificamente formattato come *TDecisionQuery*. Per una panoramica su come i componenti di supporto decisionale gestiscono e ordinano questi dati, consultare la pagina 20-1.

Per impostazione predefinita, la prima dimensione della riga appare come l’asse delle x e la prima dimensione della colonna appare come l’asse delle y.

È possibile usare i grafici decisionali in alternativa o in aggiunta alle griglie decisionali, che presentano dati incrociati in forma tabellare. Le griglie decisionali e i grafici decisionali che sono collegati alla stessa sorgente decisionale presentano le stesse dimensioni di dati. Per mostrare i diversi dati di riepilogo per le stesse dimensioni, è possibile collegare più di un grafico decisionale alla stessa sorgente decisionale. Per mostrare dimensioni diverse, collegare i grafici decisionali a sorgenti decisionali diverse.

Nella Figura 20.4, ad esempio, il primo pivot decisionale e il primo grafico sono collegati alla prima sorgente decisionale, mentre il secondo pivot decisionale e il secondo grafico sono collegati alla seconda. In questo modo, ciascun grafico mostra dimensioni diverse.

Figura 20.4 Grafici decisionali collegati a sorgenti decisionali diverse



Per maggiori informazioni su ciò che appare in un grafico decisionale, consultare la prossima sezione "Visualizzazione del grafico decisionale".

Per creare un grafico decisionale, consultare la sezione precedente "Creazione di grafici decisionali".

Per una trattazione delle proprietà del grafico decisionale e delle modalità per modificare l'aspetto e il comportamento dei grafici decisionali, consultare "Personalizzazione dei grafici decisionali" a pagina 20-16.

Visualizzazione del grafico decisionale

Per impostazione predefinita, il grafico decisionale traccia i valori di riepilogo delle categorie nel primo campo attivo della riga (lungo l'asse delle y) rispetto ai valori nel primo campo attivo della colonna (lungo l'asse delle x). Ogni categorie riportata nel grafico appare come una serie separata.

Se viene selezionata una sola dimensione—ad esempio, facendo clic solamente su un pulsante di *TDecisionPivot*—viene riportata in grafico una sola serie.

Se si è utilizzato un pivot decisionale, è possibile premerne i pulsanti per determinare quali campi del cubo decisionale (dimensioni) devono essere riportati in grafico. Per scambiare gli assi del grafico, trascinare i pulsanti di dimensione del pivot decisionale da un lato all'altro dello spazio separatore. Se si ha un grafico mono-dimensionale con tutti i pulsanti disposti su un solo lato dello spazio separatore, è possibile usare l'icona Row o l'icona Column per aggiungere pulsanti sull'altro lato del separatore e trasformare il grafico in multidimensionale.

Per attivare una sola riga e una sola colonna alla volta, è possibile impostare la proprietà *ControlType* di *TDecisionSource* a *xtRadio*. A quel punto, ci può essere un solo campo attivo alla volta per ciascun asse del cubo decisionale, mentre la funzionalità del pivot decisionale corrisponderà al comportamento del grafico. *xtRadioEx* funziona in modo analogo a *xtRadio*, ma non prevede la condizione in cui tutte le dimensioni della riga o della colonna siano chiuse.

Quando la griglia e il grafico decisionali sono connessi allo stessa *TDecisionSource*, sarà probabilmente opportuno reimpostare *ControlType* a *xtCheck* in modo che corrisponda al comportamento più flessibile di *TDecisionGrid*.

Personalizzazione dei grafici decisionali

Il componente grafico decisionale, *TDecisionGraph*, visualizza i campi provenienti dalla sorgente decisionale (*TDecisionSource*) come un grafico dinamico che si modifica quando le dimensioni dei dati vengono aperte, chiuse, spostate con “drag-and-drop”, o quando vengono riordinate usando il pivot decisionale (*TDecisionPivot*). È possibile modificare tipo, colori, tipi di contrassegni per le linee del grafico e molte altre proprietà dei grafici decisionali.

Per personalizzare un grafico:

1. Fare clic col tasto destro sul grafico e scegliere Edit Chart. Apparirà la finestra di dialogo Chart Editing.
2. Usare la pagina Chart della finestra di dialogo Chart Editing per visualizzare un elenco di serie visibili, selezionare la definizione della serie da usare quando, per una stessa serie, ve ne sono due o più disponibili, modificare i tipi di grafico per un modello o per una serie, e impostare le proprietà complessive del grafico.

   L’elenco Series sulla pagina Chart mostra tutte le dimensioni del cubo decisionale (precedute da Template:) e le categorie attualmente visibili. Ciascuna categoria o serie è un oggetto separato. È possibile:

   - Aggiungere o cancellare serie derivare da serie esistenti del grafico decisionale. Le serie derivate sono in grado di fornire annotazioni su serie esistenti o rappresentare valori calcolati da altre serie.
   - Modificare il tipo di grafico predefinito e il titolo dei modelli e delle serie.

   Per una descrizione degli altri separatori della pagina Chart cercare l’argomento “Chart page (Chart Editing dialog box)” nella Guida in linea.
3. Usare la pagina Series per impostare i modelli delle dimensioni, e personalizzare poi le proprietà per ogni singola serie del grafico.

Per impostazione predefinita, tutte le serie vengono tracciate come grafici a barre a cui vengono assegnati fino a 16 colori. È possibile modificare il tipo di modello e le proprietà per creare una nuova impostazione predefinita. Poi, mentre si agisce sulla sorgente decisionale facendola passare a stati differenti, il modello viene usato per creare dinamicamente le serie per ciascun nuovo stato. Per maggiori dettagli sui modelli, consultare “Impostazioni predefinite dei modelli di grafico decisionale” a pagina 20-17.

Per personalizzare le singole serie, seguire le istruzioni in "Personalizzazione delle serie del grafico decisionale" a pagina 20-18.

Per una descrizione dei separatori della pagina Series, cercare l'argomento "Series page (Chart Editing dialog box)" nella Guida in linea.

Impostazioni predefinite dei modelli di grafico decisionale

I grafici decisionali visualizzano i valori provenienti da due dimensioni del cubo decisionale: una dimensione è visualizzata come un asse del grafico, l'altra è usata per creare un insieme di serie. Il modello di quella dimensione fornisce le proprietà predefinite per quelle serie (per esempio, se le serie sono barre, linee, aree e così via). Via via che gli utenti passano da uno stato all'altro, tutte le serie richieste per la dimensione vengono create utilizzando il tipo di serie e le altre impostazioni predefinite specificate nel modello.

Per i casi in cui gli utenti passano ad uno stato con una sola dimensione attiva, viene fornito un modello separato. Un stato monodimensionale viene spesso rappresentato con un grafico a torta, e quindi, per questo caso, viene fornito un modello a parte.

È possibile

- Modificare il tipo di grafico predefinito.
- Modificare altre proprietà del modello grafico.
- Visualizzare e impostare le proprietà complessive del grafico.

Modifica del tipo di grafico decisionale predefinito

Per modificare il tipo di grafico predefinito:

1. Selezionare un modello nell'elenco Series della pagina Chart della finestra di dialogo Chart Editing.
2. Fare clic sul pulsante Change.
3. Selezionare un nuovo tipo e chiudere la finestra di dialogo Gallery.

Modifica di altre proprietà del modello di grafico decisionale

Per modificare il colore o le altre proprietà di un modello:

1. Selezionare la pagina Series nella parte superiore della finestra di dialogo Chart Editing.
2. Scegliere un modello nell'elenco a discesa nella parte superiore della pagina.
3. Scegliere l'opportuno separatore proprietà e selezionare le impostazioni.

Visualizzazione delle proprietà complessive del grafico decisionale

Per vedere e impostare proprietà del grafico decisionale diverse dal tipo e dalle serie:

1. Selezionare la pagina Chart nella parte superiore della finestra di dialogo Chart Editing.
2. Scegliere l'opportuno separatore proprietà e selezionare le impostazioni.

Personalizzazione delle serie del grafico decisionale

I modelli forniscono molte impostazioni predefinite per ciascuna dimensione del cubo decisionale, quali, ad esempio, i tipi di grafico e il modo in cui le serie vengono visualizzate. Le altre impostazioni predefinite, quale, ad esempio, il colore della serie, sono definite da *TDecisionGraph*. Volendo, è possibile sostituire le impostazioni predefinite di ciascuna serie.

I modelli sono stati ideati per i casi in cui il programma debba creare serie per le categorie, man mano che queste si rendono necessarie, ed eliminarle quando non lo sono più. Volendo, è possibile impostare serie personalizzate per determinati valori della categoria. Per fare ciò occorre agire sul grafico in modo che la vista attuale presenti una serie per la categoria che si vuole personalizzare. Quando la serie è visualizzata sul grafico, si può usare il Chart editor per:

- Modificare il tipo di grafico.
- Modificare le altre proprietà della serie.
- Salvare le serie del grafico che sono state personalizzate.

Per definire i modelli della serie e impostare le impostazioni predefinite complessive del grafico, consultare la sezione “Impostazioni predefinite dei modelli di grafico decisionale” a pagina 20-17.

Modifica del tipo di grafico delle serie

Per impostazione predefinita, ciascuna serie ha lo stesso tipo di grafico, definito dal modello per la sua dimensione. Per modificare tutte le serie e fare in modo che abbiano tutte lo stesso tipo di grafico, è possibile modificare il tipo di modello. Per istruzioni consultare il paragrafo “Modifica del tipo di grafico decisionale predefinito” a pagina 20-17.

Per modificare il tipo di grafico per una singola serie:

1. Selezionare una serie nell’elenco Series della pagina Chart di Chart editor.
2. Fare clic sul pulsante Change.
3. Selezionare un nuovo tipo e chiudere la finestra di dialogo Gallery.
4. Spuntare la casella di controllo Save Series.

Modifica di altre proprietà delle serie del grafico decisionale

Per modificare il colore o le altre proprietà di una serie di un grafico decisionale:

1. Selezionare la pagina Series nella parte superiore della finestra di dialogo Chart Editing.
2. Scegliere una serie nell’elenco a discesa nella parte superiore della pagina.
3. Scegliere l’opportuno separatore proprietà e selezionare le impostazioni.
4. Spuntare la casella di controllo Save Series.

Salvataggio delle impostazioni delle serie del grafico decisionale

Per impostazione predefinita, solamente le impostazioni dei modelli vengono salvate in fase di progetto. Le modifiche apportate a una determinata serie vengono salvate solo se la casella Save è attiva per quella serie nella finestra di dialogo Chart Editing.

Il salvataggio delle serie può essere dispendioso in termini di memoria, quindi, se non occorre salvarle si può disattivare la casella Save.

Componenti di supporto decisionale in esecuzione

Durante l'esecuzione, gli utenti possono compiere molte operazioni facendo clic col tasto sinistro, destro e trascinando i componenti di supporto decisionale visibili. Queste operazioni, descritte nelle sezioni precedenti di questo capitolo, sono riassunte di seguito.

Pivot decisionali in esecuzione

Gli utenti possono:

- Fare clic con il tasto sinistro sul pulsante di riepilogo all'estremità sinistra del pivot decisionale per visualizzare l'elenco di riepiloghi disponibili. Possono usare questo elenco per modificare i dati di riepilogo visualizzati in griglie e grafici decisionali.
- Fare clic col tasto destro su un pulsante di dimensione e scegliere di:
  - Spostarlo dall'area della riga all'area della colonna o viceversa.
  - Selezionare Drill In per visualizzare i dati nel dettaglio.
- Fare clic con il tasto sinistro su uno dei pulsanti di dimensione dopo il comando Drill In e scegliere:
  - Open Dimension per ritornare al livello superiore di quella dimensione.
  - All Values per passare dalla visualizzazione dei soli riepiloghi alla visualizzazione dei riepiloghi e di tutti gli altri valori nelle griglie decisionali.
  - Da un elenco di categorie disponibili per quella dimensione, selezionare una categoria da sondare per avere i valori nel dettaglio.
- Fare clic con il tasto sinistro su un pulsante di dimensione per aprire o chiudere quella dimensione.
- Trascinare e rilasciare i pulsanti di dimensione dall'area della riga all'area della colonna e viceversa; possono essere rilasciati accanto ai pulsanti esistenti in quell'area o sull'icona della riga o della colonna.

Griglie decisionali in esecuzione

Gli utenti possono:

- Fare clic col tasto destro all'interno della griglia decisionale e scegliere di:
  - Attivare o disattivare i totali parziali per singoli gruppi di dati, per tutti i valori di una dimensione o per la griglia intera.
  - Visualizzare il Decision Cube editor, descritto a pagina 20-7.
  - Aprire e chiudere le dimensioni e i riepiloghi.
- Fare click su + e su - all'interno delle intestazioni di riga e di colonna per aprire e chiudere le dimensioni.
- Trascinare e rilasciare le dimensioni dalle righe alle colonne e viceversa.

Grafici decisionali in esecuzione

Gli utenti possono trascinare le dimensioni da lato all'altro o in su e in giù nell'area della griglia del grafico per fare scorrere sullo schermo categorie e valori.

Componenti di supporto decisionale e controllo della memoria

Il caricamento di una dimensione o di un riepilogo nel cubo decisionale richiede parecchia memoria. L'aggiunta di un nuovo riepilogo incrementa il consumo di memoria in modo lineare: ovvero, un cubo decisionale che dispone di due riepiloghi utilizza il doppio della memoria che lo stesso userebbe con un riepilogo solo; un cubo decisionale con tre riepiloghi usa il triplo della memoria che lo stesso userebbe con un solo riepilogo e così via. L'aumento del consumo della memoria per le dimensioni è ancora più rapido. L'aggiunta di una dimensione con 10 valori aumenta il consumo di memoria di dieci volte. L'aggiunta di una dimensione con 100 valori aumenta il consumo di memoria di 100 volte. Quindi, l'aggiunta di dimensioni a un cubo decisionale può avere un effetto drammatico sull'uso della memoria, e può causare rapidamente problemi di prestazioni. Questo effetto è particolarmente evidente quando si aggiungono dimensioni che hanno molti valori.

I componenti di supporto decisionale dispongono di un certo numero di impostazioni che aiutano l'utente a controllare come e quando la memoria viene usata. Per maggiori informazioni sulle proprietà e sulle tecniche descritte in questa sezione, consultare *TDecisionCube* nella Guida in linea.

Impostazione dei valori massimi per dimensioni, riepiloghi e celle

Le proprietà *MaxDimensions* e *MaxSummaries* del cubo decisionale possono essere usate con la proprietà *CubeDim->ActiveFlag* per controllare il numero di dimensioni e di riepiloghi che è possibile caricare in una sola volta.È possibile impostare i valori massimi sulla pagina Cube Capacity del Decision Cube editor per introdurre un controllo complessivo sul numero di dimensioni o di riepiloghi da portare contemporaneamente in memoria.

Limitando il numero delle dimensioni o dei riepiloghi, si ottiene un calcolo approssimativo della quantità di memoria usata dal cubo decisionale. Comunque,

non è possibile distinguere tra dimensioni con molti valori e dimensioni con pochi valori. Per avere un controllo più preciso dello spazio assoluto di memoria richiesto dal cubo decisionale, è possibile limitare anche il numero delle celle nel cubo. Impostare il numero massimo di celle sulla pagina Cube Capacity del Decision Cube editor.

Impostazione dello stato della dimensione

La proprietà *ActiveFlag* controlla quali dimensioni vengono caricate. È possibile impostare questa proprietà sul separatore Dimension Settings del DecisionCube editor utilizzando il controllo Activity Type. Quando questo controllo è impostato ad *Active*, non ci sono limiti nel caricamento della dimensione che quindi occuperà sempre più spazio. Si noti che il numero di dimensioni in questa condizione deve sempre essere inferiore a quello di *MaxDimensions*, e il numero dei riepiloghi impostati ad *Active* deve essere inferiore a quello di *MaxSummaries*. Si dovrebbe impostare una dimensione o un riepilogo ad *Active* solo se è indispensabile che siano sempre disponibili. Un'impostazione *Active* diminuisce la capacità del cubo di gestire la memoria disponibile.

Quando *ActiveFlag* è impostato a *AsNeeded*, una dimensione o un riepilogo vengono caricati solamente se è possibile farlo senza superare i limiti di *MaxDimensions*, *MaxSummaries* o di *MaxCells*. Il cubo decisionale gestirà le dimensioni e i riepiloghi contrassegnati da *AsNeeded* eseguendo swap di memoria per caricarli e scaricarli, in modo da rimanere nei limiti imposti da *MaxCells*, *MaxDimensions* e *MaxSummaries*. In questo modo, una dimensione o un riepilogo non possono essere caricati in memoria se in quel momento non sono utilizzati. L'impostazione delle dimensioni che non vengono usate di frequente, come *AsNeeded*, comporta un miglioramento delle prestazioni in fase di caricamento e di gestione, anche se l'accesso alle dimensioni non caricate risulterà leggermente più lento.

Utilizzo delle dimensioni paginate

Quando Binning è impostato a Set nella pagina Dimension Settings del Decision Cube editor e Start Value non è NULL, la dimensione si dice essere "paginata" o "permanentemente espansa". È possibile accedere ai dati solo per un singolo valore alla volta di quella dimensione, sebbene da programma sia possibile accedere a una serie di valori in modo sequenziale. Tale dimensione non può essere usata come pivot o aperta.

È estremamente dispendioso, dal punto di vista della memoria, includere dati dimensionali per le dimensioni che hanno un numero elevato di valori. Paginando tali dimensioni, è possibile visualizzare informazioni di riepilogo per un valore alla volta. Solitamente, quando vengono visualizzate in questo modo, le informazioni sono più facili da leggere e il consumo di memoria è molto più semplice da gestire.

Capitolo

21

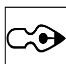
Connessione ai database

La maggior parte dei componenti dataset può collegarsi direttamente a un server di database. Appena connesso, il dataset comunica con il server automaticamente. Quando si apre il dataset, esso si popola con i dati dal server e quando si registrano i record, essi vengono restituiti al server e applicati. Un singolo componente connection SQL può essere condiviso da più dataset oppure ciascun dataset può utilizzare la propria connessione.

Ogni tipo del dataset si collega al server di database utilizzando il proprio tipo di componente connection, progettato per funzionare con un singolo meccanismo di accesso ai dati. La seguente tabella elenca questi meccanismi di accesso ai dati e i componenti connection associati:

Tabella 21.1 Componenti di connessione del database

Meccanismo di accesso ai dati	Componente connection
Borland Database Engine (BDE)	TDatabase
ActiveX Data Objects (ADO)	TADOConnection
dbExpress	TSQLConnection
InterBase Express	TIBDatabase



Per una trattazione dei pro e contro di ciascuno di questi meccanismi consultare "Uso dei databases" a pagina 18-1.

Il componente connection fornisce tutte le informazioni necessarie per stabilire una connessione al database. Queste informazioni sono diverse per ogni tipo di componente connection:

- Per informazioni sulla descrizione di una connessione basata su BDE, vedere "Identificazione del database" a pagina 24-14.
- Per informazioni sulla descrizione di una connessione basata su ADO, vedere "Connessione a un datastore mediante TADOConnection" a pagina 25-3.

- Per informazioni sulla descrizione di una connessione *dbExpress*, vedere "Impostazione di TSQLConnection" a pagina 26-3.
- Per informazioni sulla descrizione di una connessione InterBase Express, vedere la Guida in linea per *TIBDatabase*.

Benché ogni tipo di dataset utilizzi un componente connection diverso, tali componenti discendono tutti da *TCustomConnection*. Tutti quanti assolvono moltissimi compiti identici e rendono disponibili molte proprietà, metodi ed eventi identici. Il presente capitolo tratta la maggior parte di queste attività comuni.

Uso di connessioni implicite

Indipendentemente dal meccanismo di accesso ai dati utilizzato, è sempre possibile creare il componente connection in modo esplicito e utilizzarlo per gestire la connessione a un server di database e la comunicazione con esso. Per dataset BDE e basati su ADO, c'è la possibilità di descrivere la connessione al database tramite proprietà del dataset e di permettere al dataset di generare una connessione implicita. Per i dataset BDE, si specifica una connessione implicita utilizzando la proprietà *DatabaseName*. Per i dataset basati su ADO, si utilizza la proprietà *ConnectionString*.

Quando si utilizza una connessione implicita, non è necessario creare esplicitamente un componente connection. Ciò può semplificare lo sviluppo dell'applicazione e la connessione predefinita che si specifica può coprire un'ampia gamma di situazioni. Per applicazioni client/server complesse e cruciali, con molti utenti e diversi requisiti per connessioni al database, comunque, si consiglia di creare i propri componenti connection in modo da adattare ciascuna connessione di database alle necessità dell'applicazione. I componenti connection espliciti offrono un maggiore controllo. Ad esempio, è necessario accedere al componente connection per eseguire le seguenti operazioni:

- Personalizzare il supporto per il login al server di database. (le connessioni implicite visualizzano una finestra di dialogo di login predefinita per richiedere all'utente un nome utente e una password.)
- Controllare le transazioni e specificarne i livelli di isolamento delle transazioni.
- Eseguire comandi SQL sul server senza utilizzare un dataset.
- Compiere operazioni su tutti i dataset aperti che sono connessi allo stesso database.

Inoltre, se si hanno più dataset che utilizzano tutti lo stesso server, potrebbe essere più semplice utilizzare un componente connection, in modo che si debba specificare una sola volta il server da utilizzare. In questo modo, se in un secondo momento si modificherà il server, non sarà necessario aggiornare tutti i componenti dataset ma solo il componente connection.

Controllo delle connessioni

Prima di potere stabilire una connessione a un server di database, l'applicazione deve fornire determinate informazioni chiave che descrivono il server desiderato. Ogni tipo di componente connection rende disponibile un diverso set di proprietà per permettere l'identificazione del server. Di solito, comunque, offrono tutti un modo per assegnare il nome del server desiderato e per fornire un set di parametri di connessione che controllano il modo in cui viene stabilita la connessione. I parametri di connessione variano da server a server. Possono includere informazioni come nome utente e password, dimensione massima dei campi BLOB, ruoli SQL, e così via.

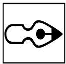
Una volta identificati il server richiesto e tutti i parametri di connessione, è possibile utilizzare il componente connection per aprire o chiudere esplicitamente una connessione. All'apertura e alla chiusura di una connessione, il componente connection genera eventi che è possibile utilizzare per personalizzare la risposta dell'applicazione alle modifiche nella connessione al database.

Connessione a un server di database

Esistono due modi per connettersi a un server database usando un componente connection:

- Chiamando il metodo *Open*.
- Impostando la proprietà *Connected* a **true**.

La chiamata del metodo *Open* imposta *Connected* a **true**.

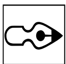


Quando un componente connection non è connesso a un server e un'applicazione tenta di aprire i dataset a lui associati, il dataset chiama automaticamente il metodo *Open* del componente connection.

Quando si imposta *Connected* a **true**, il componente connection genera per prima cosa un evento *BeforeConnect*, in cui è possibile eseguire qualsiasi inizializzazione. Ad esempio, è possibile utilizzare questo evento per modificare i prametri di connessione.

Dopo l'evento *BeforeConnect*, il componente connection può visualizzare una finestra di dialogo predefinita per il login, a seconda di come si sceglie di controllare il login al server. Quindi passa al driver il nome utente e la password, aprendo una connessione.

Una volta aperta la connessione, il componente connection genera un evento *AfterConnect* in cui è possibile eseguire qualsiasi operazione che richieda una connessione aperta.



Alcuni componenti connection generano anche altri eventi nel momento in cui stabiliscono una connessione.

Una volta che la connessione è stata stabilita, tale connessione viene mantenuta attiva finché c'è almeno un dataset attivo che la utilizza. Quando non ci sono più dataset attivi, il componente connection rilascia la connessione. Alcuni componenti connection rendono disponibile una proprietà *KeepConnection* che permette alla

connessione di rimanere aperta anche se tutti i dataset che la utilizzano sono chiusi. Se *KeepConnection* è impostato a **true**, la connessione viene mantenuta aperta. Per connessioni a server di database remoti o per applicazioni che aprono e chiudono dataset con una certa frequenza, l'impostazione di *KeepConnection* a **true** riduce il traffico di rete e velocizza l'applicazione. Se *KeepConnection* è impostato a false, la connessione viene interrotta appena non vi sono più dataset attivi che utilizzano il database. Se un dataset che utilizza il database viene aperto in un secondo momento, la connessione deve essere ristabilita e inizializzata.

Disconnessione da un server di database

Esistono due modi per sconnettersi da un server usando un componente connection:

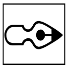
- Impostando la proprietà *Connected* a **false**.
- Chiamata al metodo *Close*.

La chiamata a *Close* imposta *Connected* a **false**.

Quando la proprietà *Connected* è impostata a **false**, il componente connection genera un evento *BeforeDisconnect* in cui è possibile eseguire qualsiasi operazione di pulizia prima che la connessione venga chiusa. Ad esempio, è possibile utilizzare questo evento per memorizzare nella cache le informazioni su tutti i dataset aperti prima che essi vengano chiusi.

Dopo l'evento *BeforeConnect*, il componente connection chiude tutti i dataset aperti e si scollega dal server.

Infine, il componente connection genera un evento *AfterDisconnect* in cui è possibile rispondere alla modifica nello stato della connessione, come ad esempio l'attivazione di un pulsante Connect dell'interfaccia utente.



La chiamata al metodo *Close* o l'impostazione della proprietà *Connected* a **false** causa lo scollegamento da un server di database anche se la proprietà *KeepConnection* del componente connection è impostata a **true**.

Controllo del login al server

La maggior parte dei server database remoti includono funzioni di sicurezza per proibire l'accesso non autorizzato. Generalmente, il server richiede un nome utente e una password di login prima di consentire l'accesso al database.

In fase di progettazione, se un server richiede un login, C++Builder visualizza una finestra di dialogo Login standard che, quando si tenta di connettersi al database per la prima volta, richiede il nome dell'utente e la password.

In fase di esecuzione, ci sono tre modi con cui gestire la richiesta di login del server:

- Consentire la gestione del login alla finestra di dialogo di login e di elaborazione. Questo è il metodo predefinito. Impostare la proprietà *LoginPrompt* del componente connection a **true** (impostazione predefinita) e aggiungere DBLogDlg.hpp alla clausola che dichiara il componente connection.

L'applicazione visualizzerà la finestra di dialogo di login standard appena il server richiede un nome utente e una password.

- Fornire le informazioni di login prima del tentativo di login. Ogni tipo di componente connection utilizza un meccanismo diverso per la specifica del nome utente e della password:
  - Per i dataset BDE, *dbExpress* e InterBaseExpress, si può accedere ai parametri di connessione nome utente e password tramite la proprietà *Params*. (Per i dataset BDE, i valori dei parametri possono essere associati anche a un alias BDE, mentre per i dataset dbExpress possono essere associati anche a un nome di connessione).
  - Per i dataset ADO, il nome utente e la password possono essere inclusi nella proprietà *ConnectionString* (o forniti come parametri al metodo *Open*).

  Se si specificano nome utente e password prima che il server li richieda, accertarsi di impostare il *LoginPrompt* a **false**, in modo che non venga visualizzata la finestra di dialogo login predefinita. Ad esempio, il codice seguente imposta il nome e la password dell'utente per un componente SQl connection nel gestore di evento *BeforeConnect*, decifrando una password crittografata memorizzata associata al nome di connessione corrente:

```
void __fastcall TForm1::SQLConnectionBeforeConnect(TObject *Sender)
{
  if (SQLConnection1->LoginPrompt == false)
  {
    SQLConnection1->Params->Values["User_Name"] = "SYSDBA";
    SQLConnection1->Params->Values["Password"] =
        Decrypt(SQLConnection1->Params->Values["Password"]);
  }
}
```

  Si noti che l'impostazione del nome utente e della password in fase di progettazione o l'utilizzo di stringhe direttamente nel codice fa sì che i valori vengano incorporati nel file eseguibile dell'applicazione. In questo modo, i valori sono ancora abbastanza facilmente reperibili, compromettendo in tal modo la sicurezza del server.

- Preparare la propria gestione custom per l'evento di login. Il componente connection genera un evento quando ha bisogno del nome utente e della password.
  - Per *TDatabase*, *TSQLConnection* e *TIBDatabase* questo è un evento *OnLogin*. Il gestore di evento ha due parametri, il componente connection e una copia locale dei parametri nome utente e password in una lista di stringhe. (*TSQLConnection* include anche il parametro database). Perché questo evento si verifichi, è necessario impostare la proprietà *LoginPrompt* a **true**. Un valore di *LoginPrompt* impostato a **false** e l'assegnazione di un gestore all'evento *OnLogin* crea una situazione in cui è impossibile connettersi al database perché la finestra di dialogo predefinita non viene visualizzata e il gestore di evento *OnLogin* non sarà mai eseguito.
  - Per *TADOConnection*, l'evento è un evento *OnWillConnect*. Il gestore di evento ha cinque parametri, il componente connection e quattro parametri che

restituiscono valori per influire sulla connessione (di cui due per nome utente e password). Questo evento si verifica sempre, indipendentemente dal valore di *LoginPrompt*.

Scrivere un gestore di evento per l'evento in cui si impostano i parametri di login. Di seguito è riportato un esempio in cui i valori per i parametri USER NAME e PASSWORD sono forniti mediante una variabile globale (*UserName*) e un metodo che, a fronte di un nome utente (*PasswordSearch*), restituisce una password)

```
void __fastcall TForm1::Database1Login(TDatabase *Database, TStrings *LoginParams)
{
  LoginParams->Values["USER NAME"] = UserName;
  LoginParams->Values["PASSWORD"] = PasswordSearch(UserName);
}
```

Come già visto con gli altri metodi utilizzati per fornire i parametri di login, quando si scrive un gestore di evento *OnLogin* o *OnWillConnect* è bene evitare di scrivere esplicitamente la password nel codice dell'applicazione. La password dovrebbe apparire solo come valore crittografato, come un elemento in un database sicuro utilizzato dall'applicazione per cercare il valore, oppure dovrebbe essere ottenuta dinamicamente dall'utente.

Gestione delle transazioni

Una *transazione* è un gruppo di azioni che devono essere tutte portate a termine con successo su una o più tabelle di un database prima che siano *compiute* (rese permanenti). Se una delle azioni nel gruppo non riesce, allora tutte le azioni *vengono fatte scorrere a ritroso* (annullate). Utilizzando le transazioni, nel caso si verifichi un problema durante il compimento di una delle azione che compongono la transazione, ci si assicura che il database non venga lasciato in uno stato incongruente.

Ad esempio, in un'applicazione bancaria, il trasferimento di fondi da un conto all'altro è un'operazione che sarebbe bene proteggere con una transazione. Se, dopo aver decrementato il saldo di un conto, si verifica un errore incrementando il saldo dell'altro, sarà possibile annullare la transazione in modo che il database rifletta ancora il saldo totale corretto.

È sempre possibile gestire le transazioni inviando comandi SQL direttamente al database. Quasi tutti i database sono dotati di un proprio modello per la gestione delle transazioni, benché alcuni non forniscano alcun supporto per le transazione. Per i server che le supportano, potrebbe essere necessario scrivere in prima persona il codice per la gestione delle transazioni, sfruttando le capacità di gestione di transazioni avanzate esistenti su un particolare server di database, come la cache di schema.

Se non è necessario utilizzare queste capacità evolute di gestione delle transazioni, i componenti connection mettono a disposizione un insieme di metodi e proprietà che è possibile utilizzare per gestire le transazioni senza inviare esplicitamente alcun comando SQL. L'utilizzo di queste proprietà e di questi metodi offre il vantaggio che non è necessario personalizzare l'applicazione per ogni tipo di server di database

utilizzato, sempre che il server supporti le transazioni. (Anche BDE fornisce un supporto limitato per le transazione con tabelle locali che non dispongono di supporto per transazioni del server). (Se non si utilizza BDE il tentativo di avviare le transazioni su un database che non le supporta provoca il sollevamento di un'eccezione da parte dei componenti connection.)



Quando un componente dataset provider applica gli aggiornamenti, genera implicitamente transazioni per qualsiasi aggiornamento. È bene prestare attenzione al fatto che qualsiasi transazione iniziata esplicitamente non vada in conflitto con quelle generati dal provider.

Avvio di una transazione

Quando si avvia una transazione, tutte le successive istruzioni di lettura o di scrittura nel database si verificano nel contesto di quella transazione, finché la transazione non viene terminata esplicitamente o, nel caso di transazioni sovrapposte, finché non viene avviata un'altra transazione. Ogni istruzione è considerata parte di un gruppo. Tutte le modifiche dovranno essere registrate con successo nel database, altrimenti sarà necessario annullare ogni modifica fatta nel gruppo.

Mentre la transazione è in corso, la vista dei dati nelle tabelle di database è determinata dal livello di isolamento della transazione. Per ulteriori informazioni sui livelli di isolamento delle transazioni, consultare "Specifica del livello di isolamento delle transazioni" a pagina 21-10.

Per *TADOConnection* avviare una transazione chiamando il metodo *BeginTrans*:

```
Level = ADOConnection1->BeginTrans();
```

BeginTrans restituisce il livello di annidamento della transazione appena iniziata. Una transazione annidata è una transazione inclusa all'interno di un'altra transazione genitore. Dopo che il server ha avviato la transazione, la connessione ADO riceve un evento *OnBeginTransComplete*.

Per *TDatabase*, utilizzare il metodo *StartTransaction*method invece.*TDataBase* non supporta transazioni annidate o sovrapposte: Se si chiama il metodo *StartTransaction* di un componente *TDatabase* mentre è in corso un'altra transazione, il componente solleva un'eccezione. Per evitare di chiamare *StartTransaction*, è possibile controllare la proprietà *InTransaction* property:

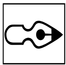
```
if (!Database1->InTransaction)
  Database1->StartTransaction();
```

Anche *TSQLConnection* utilizza il metodo *StartTransaction* ma la versione utilizza offre un controllo maggiore. Nello specifico, *StartTransaction* accetta un descrittore di transazione che permette di gestire più transazioni simultanee e di specificare il livello di isolamento delle transazioni, transazione per transazione. (Per ulteriori informazioni sui livelli delle transazioni, vedere "Specifica del livello di isolamento delle transazioni" a pagina 21-10.) Per gestire più transazioni simultanee, impostare il campo *TransactionID* del descrittore di transazione a un valore univoco. *TransactionID* può essere un qualsiasi valore scelto dall'utente, a patto che sia univoco (non sia cioè in conflitto con una qualsiasi altra transazione attualmente in

esecuzione). A seconda del server, le transazioni avviate da *TSQLConnection* possono essere annidate (come quando si utilizza ADO) o sovrapposte.

```
TTransactionDesc TD;
TD.TransactionID = 1;
TD.IsolationLevel = xilREADCOMMITTED;
SQLConnection1->StartTransaction(TD);
```

Per impostazione predefinita, nel caso di transazioni sovrapposte, la prima transazione diventa inattiva non appena parte la seconda transazione, benché sia possibile rimandare la conferma o l'annullamento della prima transazione ad una fase successiva. Se si utilizza *TSQLConnection* con un database InterBase, è possibile identificare ogni dataset nell'applicazione con una particolare transazione attiva, impostandone la proprietà *TransactionLevel*. Ciò significa che, dopo avere avviato una seconda transazione, è possibile continuare a gestire contemporaneamente entrambe le transazioni, semplicemente associando un dataset alla transazione che si desidera.



A differenza di *TADOConnection*, *TSQLConnection* e *TDatabase* non ricevono alcun evento quando parte la transazione.

InterBase Express offre un controllo ancora maggiore rispetto a quello di *TSQLConnection* perché utilizza un componente di transazione separato invece di un singolo componente connection che avvia anche le transazioni. È possibile utilizzare, tuttavia, *TIBDatabase* per avviare una transazione predefinita:

```
if (!IBDatabase1->DefaultTransaction->InTransaction)
  IBDatabase1->DefaultTransaction->StartTransaction();
```

È possibile ottenere transazioni sovrapposte utilizzando due componenti di transazione separati. Ogni componente di transazione ha un set di parametri che permettono di configurare la transazione. Questi parametri permettono di specificare il livello di isolamento della transazione, oltre ad altre proprietà.

Chiusura di una transazione

Idealmente, una transazione dovrebbe durare solo il tempo necessario. Più a lungo una transazione è attiva, maggiore è il numero di utenti che accedono simultaneamente al database, e più è grande il numero di transazioni simultanee che vengono avviate e terminano durante l'esistenza della transazione, maggiore è la probabilità che la transazione sia in conflitto con un'altra quando si tenterà di confermare una qualsiasi modifica.

Chiusura di una transazione andata a buon fine

Non appena tutte le azioni che compongono la transazione sono andate a buon fine, è possibile rendere permanenti le modifiche al database confermando la transazione. Per *TDatabase*, si convalida una transazione utilizzando il metodo *Commit*:

```
MyOracleConnection->Commit();
```

Anche per *TSQLConnection* si utilizza il metodo *Commit* ma è necessario specificare quale transazione si sta convalidando fornendo lo stesso descrittore della transazione passato al metodo *StartTransaction*:

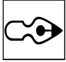
```
MyOracleConnection->Commit(TD);
```

Per *TIBDatabase*, si convalida un oggetto transazione utilizzando il metodo *Commit*:

```
IBDatabase1->DefaultTransaction->Commit();
```

Per *TADOConnection*, si convalida una transazione utilizzando il metodo *CommitTrans*:

```
ADOConnection1->CommitTrans();
```



È possibile convalidare una transazione annidata, solo affinché sia possibile annullare le le modifiche in un secono momento nel caso venga annullata la transazione genitore.

Dopo che la transazione è stata convalidata con successo, un componente connection ADO riceve un evento *OnCommitTransComplete*. Altri componenti connection non ricevono un tale evento.

Una chiamata per convalidare la transazione corrente di solito viene tentata in un blocco di istruzioni **try...catch**. In questo modo, se la transazione non viene inoltrata con esito positivo, è possibile utilizzare il blocco **catch** per gestire l'errore e ritentare l'operazione oppure per eseguire il roll-back della transazione.

Chiusura di una transazione non andata a buon fine

Nel caso si verifichi un errore durante l'esecuzione delle modifiche che fanno parte della transazione o durante la conferma della transazione, si renderà necessario eliminare tutte le modifiche apportate dalla transazione. L'abbandono delle modifiche è detto annullamento, o roll-back, della transazione.

Per *TDatabase*, si annulla una transazione chiamando il metodo *Rollback*:

```
MyOracleConnection->Rollback();
```

Per *TSQLConnection*, è possibile usaare anche il metodo *Rollback*, ma è necessario specificare quale *transazione* si sta annullando fornendo il descrittore della *transazione* passato al metodo *StartTransaction*:

```
MyOracleConnection->Rollback(TD);
```

Per *TIBDatabase*, è possibile annullare un oggetto *transaction* chiamandone il metodo *Rollback*:

```
IBDatabase1->DefaultTransaction->Rollback();
```

Per *TADOConnection*, si annulla una transazione chiamando il metodo *RollbackTrans*:

```
ADOConnection1->RollbackTrans();
```

Dopo che la transazione è stata annullata con successo, un componente ADO connection riceve un evento *OnRollbackTransComplete*. Altri componenti connection non ricevono un tale evento.

Una chiamata per annullare la transazione corrente di solito si verifica

- Nella porzione di codice che gestisce l'eccezione, nel caso sia impossibile ripristinare un errore di database.

- Nel codice dell'evento di un pulsante o di un menu, ad esempio quando l'utente fa clic su un pulsante Cancel.

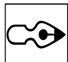
Specifica del livello di isolamento delle transazioni

Il livello di isolamento della transazione determina come una transazione interagisce con altre transazioni simultanee quando operano con le stesse tabelle. In particolare, influenza quanto una transazione "vede" delle altre modifiche apportate dalle transazioni a una tabella.

Ogni tipo di server supporta un diverso set di possibili livelli di isolamento della transazione. Esistono tre possibili livelli di isolamento di una transazione:

- *DirtyRead*: Quando il livello di isolamento è *DirtyRead*, la transazione vede tutte le modifiche fatte da altre transazioni, anche se tali modifiche non sono stati confermate. Le modifiche non confermate non sono permanenti e potrebbero essere annullate in qualsiasi momento. Questo valore fornisce il livello di isolamento minimo e non è disponibile per molti server di database (come Oracle, Sybase, MS-SQL e InterBase).
- *ReadCommitted*: Quando il livello di isolamento è *ReadCommitted*, risultano visibili solo le modifiche confermate fatte da altre transazioni. Benché questa impostazione impedisca alla transazione di vedere le modifiche non confermate e che potrebbero essere annullate, nel caso un'altra transazione venga confermata durante la fase di lettura della transazione in corso, è tuttavia possibile che si abbia una vista incongruente dello stato del database. Questo livello è disponibile per tutte le transazioni tranne che per le transazioni locali gestite da BDE.
- *RepeatableRead*: Quando il livello di isolamento è *RepeatableRead*, alla transazione viene garantita la visione di uno stato congruente dei dati del database. La transazione vede una singola istantanea dei dati. Non è in grado di vedere nessuna successiva modifica ai dati da parte di altre transazioni simultanee, anche nel caso in cui le mdifiche vengano confermate. Questo livello di isolamento garantisce che una volta che la transazione ha letto un record, la vista di quel record non cambierà. A questo livello la transazione risulta veramente isolata dalle modifiche fatte da altre transazioni. Questo livello è non disponibile su alcuni server, come Sybase e MS-SQL e non è disponibile per transazioni locali gestite da BDE.

Inoltre, *TSQLConnection* permette di specificare livelli di isolamento custom, specifici per un certo database. I livelli di isolamento custom sono definiti dal driver *dbExpress*. Consultare la documentazione del driver per i dettagli.



Per una descrizione dettagliata dell'implementazione di ogni livello di isolamento, consultare la documentazione del server.

TDatabase e *TADOConnection* permettono di specificare il livello di isolamento della transazione impostando la proprietà *TransIsolation*. Quando si imposta *TransIsolation* a un valore che non è supportato dal server di database, si ottiene il successivo livello di isolamento più elevato (se disponibile). Se non c'è alcun livello più elevato disponibile, appena si prova ad avviare una transazione il componente connection solleva un'eccezione.

Utilizzando *TSQLConnection*, il livello di isolamento della transazione è controllato dal campo *IsolationLevel* del descrittore della transazione.

Utilizzando InterBase Express, il livello di isolamento della ttransazione è controllato da un parametro della transazione.

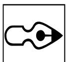
Invio di comandi al server

Tutti i componenti database connection, tranne *TIBDatabase*,permettono di eseguire istruzioni SQL sul server associato chiamando il metodo *Execute*. Benché *Execute* possa restituire un cursor nel caso l'istruzione sia un'istruzione SELECT, questo utilizzo non è consigliato. Il metodo preferenziale per l'esecuzione di istruzioni che restituiscono dati è quello di utilizzare un dataset.

Il metodo *Execute* è molto comodo per l'esecuzione di semplici istruzioni SQL che non restituiscono record. Tali istruzioni includono istruzioni Data Definition Language (DDL), che operano sui metadati di un database o li creano, come CREATE INDEX, ALTER TABLE e DROP DOMAIN. Anche alcune istruzioni DML (Data Manipulation Language) di SQL non restituiscono un set risultato. Le istruzioni DML che compiono un'azione sui dati ma non restituiscono un set risultato sono: INSERT, DELETE e UPDATE.

La sintassi del metodo *Execute* varia in funzione del tipo di connessione:

- Per *TDatabase*, *Execute* accetta quattro parametri: una stringa Ansi che specifica la singola istruzione SQL che si desidera eseguire, un oggetto *TParams* che fornisce tutti i valori dei parametri per quell'istruzione, un valore boolean che indica se l'istruzione deve essere messa in cache o meno perché la si vorrà eseguire di nuovo, e un puntatore a un cursor BDE che può essere restituito. (Si consiglia di passare il valore NULL).
- Per *TADOConnection*, esistono due versioni di *Execute*. La prima versione richiede una WideString che specifica l'istruzione SQL e un secondo parametro che specifica un set di opzioni che controllano se l'istruzione viene eseguita in modo asincrono o se restituisce dei record. Questa prima sintassi restituisce un'interfaccia per i record restituiti. La seconda sintassi richiede una WideString che specifica l'istruzione SQL, un secondo parametro che restituisce il numero di record interessati dall'esecuzione dell'istruzione, e un terzo parametro che specifica varie opzioni, come, ad esempio, se l'istruzione viene eseguita in modo asincrono. Notare che nessuna delle due sintassi consente il passaggio di parametri.
- Per *TSQLConnection*, *Execute* accetta tre parametri, una stringa ANSI che specifica la singola istruzione SQL che si desidera eseguire, un oggetto *TParams* che fornisce tutti i valori dei parametri per quell'istruzione e un puntatore che può ricevere un oggetto *TCustomSQLDataSet* creato per restituire ai record il valore NULL.



Execute può solo eseguire un'istruzione SQL alla volta. Non è possibile per eseguire più istruzioni SQL con una singola chiamata a *Execute*, come di solito si fa con i programmi di utilità di scrittura di codice SQL. Per eseguire più di un'istruzione, chiamare *Execute* più volte.

È relativamente facile eseguire un'istruzione che non include nessun parametro. Ad esempio, il seguente codice esegue un'istruzione CREATE TABLE (DDL) senza alcun parametro su un componente *TSQLConnection*:

```
void __fastcall TDataForm::CreateTableButtonClick(TObject *Sender)
{
  SQLConnection1->Connected = true;
  AnsiString SQLstmt = "CREATE TABLE NewCusts " +
    "( " +
    " CustNo INTEGER, " +
    " Company CHAR(40), " +
    " State CHAR(2), " +
    " PRIMARY KEY (CustNo) " +
    ")";
  SQLConnection1->Execute(SQLstmt, NULL, NULL);
}
```

Per utilizzare parametri, è necessario creare un oggetto *TParams*. Per ogni valore di parametro utilizzare il metodo *TParams*, per aggiungere un oggetto *TParam* utilizzare il metodo CreateParam. Quindi utilizzare le proprietà di *TParam* per descrivere il parametro e impostarne il valore.

Questo procedimento è illustrato nell'esempio seguente che utilizza *TDatabase* per eseguire un'istruzione INSERT. L'istruzione INSERT ha un singolo parametro con nome :*StateParam*. Viene creato un oggetto *TParams* (di nome *stmtParams*) per fornire il valore "CA" a quel parametro.

```
void __fastcall TForm1::INSERT_WithParamsButtonClick(TObject *Sender)
{
  AnsiString SQLstmt;
  TParams *stmtParams = new TParams;
  try
  {
    Database1->Connected = true;
    stmtParams->CreateParam(ftString, "StateParam", ptInput);
    stmtParams->ParamByName("StateParam")->AsString = "CA";
    SQLstmt = "INSERT INTO 'Custom.db' ";
    SQLstmt += "(CustNo, Company, State) ";
    SQLstmt += "VALUES (7777, 'Robin Dabank Consulting', :StateParam)";
    Database1->Execute(SQLstmt, stmtParams, false, NULL);
  }
  __finally
  {
    delete stmtParams;
  }
}
```

Se l'istruzione SQL include un parametro ma non è stato preparato un oggetto *TParam* per fornirgli un valore, l'istruzione SQL può generare un errore al momento dell'esecuzione (questo dipende dal particolare back-end di database utilizzato). Se viene fornito un oggetto *TParam* ma non c'è alcun parametro corrispondente nell'istruzione SQL, quando l'applicazione tenta di utilizzare *TParam* viene sollevata un'eccezione.

Operazioni con i dataset associati

Tutti i componenti per la connessione ai database gestiscono una lista di tutti i dataset attivi che li utilizzano per collegarsi a un database. Un componenente connection utilizza questa lista, ad esempio, per chiudere tutti i dataset nel momento in cui chiude la connessione al database.

È possibile utilizzare questa lista anche per eseguire azioni su tutti i dataset che utilizzano uno specifico componente connection per collegarsi a un particolare database.

Chiusura dei dataset senza scollegarsi dal server

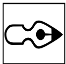
Il componente connection chiude automaticamente tutti i dataset quando si chiude la sua connessione. A volte potrebbe essere necessario chiudere tutti i dataset senza scollegarsi dal server di database.

Per chiudere tutti i dataset aperti senza scollegarsi da un server, si può usare il metodo *CloseDataSets*.

Per *TADOConnection* e per *TIBDatabase*, la chiamata a *CloseDataSets* lascia sempre aperta la connessione. Per *TDatabase* e per *TSQLConnection*, è inoltre necessario impostare la proprietà *KeepConnection* a **true**.

Scansione di tutti i dataset associati

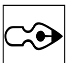
Per eseguire qualsiasi altra azione (diversa dalla chiusura) su tutti i dataset che utilizzano un componente connection, utilizzare le proprietà *DataSets* e *DataSetCount*. *DataSets* è un array con indice di tutti i dataset che sono collegati con il componente connection. Per tutti i componenti connection, tranne *TADOConnection*, questa lista include solo i dataset attivi. *TADOConnection* elenca anche i dataset non attivi. *DataSetCount* è il numero dei dataset in questo array.



Quando si utilizza un dataset client specializzato per memorizzare gli aggiornamenti in cache (diversamente dal dataset client generico, *TClientDataSet*), la proprietà *DataSets* elenca il dataset interno posseduto dal dataset client, non il dataset client stesso.

È possibile utilizzare *DataSets* con *DataSetCount* per passare in sequenza attraverso tutti i dataset correntemente attivi nel codice. Ad esempio, il seguente codice passa in sequenza tutti i dataset attivi e disabilita tutti i controlli che utilizzano i dati da essi forniti:

```
for (int i = 0; i < MyDBConnection->DataSetCount; i++)
  MyDBConnection->DataSets[i]->DisableControls();
```



Oltre ai dataset, *TADOConnection* supporta anche oggetti command. È possibile iterare tra questi oggetti esattamente come si iterano tra i dataset, utilizzando le proprietà *Commands* e *CommandCount*.

Ottenimento di metadati

Tutti i componenti database connection possono acquisire dal server di database liste di metadati, benché essi si differenzino in base al tipo di metadati che acquisiscono. I metodi per l'acquisizione di metadati riempiono una lista di stringhe con i nomi delle diverse entità disponibili sul server. È possibile utilizzare quindi queste informazioni, ad esempio, per permettere agli utenti di selezionare dinamicamente una tabella in fase di esecuzione.

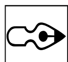
È possibile utilizzare un componente *TADOConnection* per acquisire metadati sulle tabelle e sulle procedure registrate disponibili nel datastore ADO. È possibile utilizzare quindi queste informazioni, ad esempio, per permettere agli utenti di selezionare dinamicamente una tabella o una procedura registrata in fase di esecuzione.

Elenco delle tabelle disponibili

Il metodo *GetTableNames* copia una lista di nomi di tabelle in un oggetto string list già esistente. In questo modo, ad esempio, si può riempire una casella di riepilogo con i nomi delle tabelle disponibili, utilizzabile dall'utente per scegliere una tabella da aprire. La riga seguente riempie una casella di riepilogo con i nomi di tutte le tabelle del database:

```
MyDBConnection->GetTableNames(ListBox1->Items, false);
```

GetTableNames ha due parametri: la lista di stringhe da riempire con i nomi delle tabelle, e un parametro boolean che indica se la lista deve includere le tabelle di sistema o le tabelle comuni. Si noti che non tutti i server utilizzano tabelle di sistema per memorizzare i metadati; pertanto la richiesta di tabelle di sistema potrebbe generare una lista vuota.



Per la maggior parte dei componenti database connection, se il secondo parametro è impostato a **false**, il metodo *GetTableNames* restituisce una lista di tutte le tabelle non di sistema disponibili. Per *TSQLConnection*, tuttavia, si ha un maggior controllo sui tipi di tabella aggiunti alla lista quando non si prelevano solo i nomi delle tabelle di sistema. Se si utilizza *TSQLConnection*, i tipi dei nomi aggiunti alla lista sono controllati dalla proprietà *TableScope*. *TableScope* indica se la lista dovrà contenere uno o tutti i seguenti tipi: tabelle comuni, tabelle di sistema, sinonimi e viste.

Elenco dei campi in una tabella

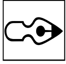
Il metodo *GetFieldNames* riempie una stringa esistente con i nomi di tutti i campi (colonne) presenti in una tabella specificata. *GetFieldNames* ha due parametri, il nome della tabella di cui si desidera elencare i campi e una lista di stringhe esistente da riempire con i nomi dei campi:

```
MyDBConnection->GetTableNames("Employee", ListBox1->Items);
```

Elenco delle procedure registrate disponibili

Per ottenere un elenco di tutte le procedure registrate presenti nel database, utilizzare il metodo *GetProcedureNames*. Questo metodo ha un singolo parametro: una lista di stringhe già esistente da riempire:

```
MyDBConnection->GetProcedureNames(ListBox1->Items);
```

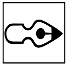


GetProcedureNames è disponibile solo per *TADOConnection* e *TSQLConnection*.

Elenco degli indici disponibili

Per ottenere un elenco di tutti gli indici di una certa tabella, utilizzare il metodo *GetIndexNames*. Questo metodo ha due parametri: la tabella di cui si desidera ottenere gli indici e una lista di stringhe già esistente da riempire:

```
MyDBConnection1->GetIndexNames("Employee", ListBox1->Items);
```



GetIndexNames è disponibile solo per *TSQLConnection*, benché la maggior parte dei dataset di tipo tabella abbia un metodo equivalente.

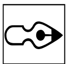
Elenco dei parametri delle procedure registrate

Per ottenere un elenco di tutti i parametri di una certa procedura registrata, utilizzare il metodo *GetProcedureParams*. *GetProcedureParams* riempie un oggetto *TList* con dei puntatori ai structure di descrizione dei parametri, in cui ogni structure descrive un parametro di una procedura registrata specificata, inclusi nome, indice, tipo di parametro, tipo di campo, e così via.

GetProcedureParams ha due parametri: il nome della procedura registrata e un oggetto *TList* già esistente da riempire:

```
MyDBConnection1->GetIndexNames("GetInterestRate", List1);
```

Per convertire queste descrizioni dei parametri aggiunte alla lista in un oggetto *TParams* più familiare chiamare la procedura globale *LoadParamListItems*procedure. Poiché *GetProcedureParams* alloca dinamicamente i singoli record, l'applicazione deve liberarli una volta che ha recuperato le informazioni. La routine globale *FreeProcParams* può farlo automaticamente.



GetProcedureParams è disponibile solo per *TSQLConnection*.

Capitolo

22

I dataset

L'unità fondamentale per accedere ai dati è la famiglia di oggetti dataset. Le applicazioni usano i dataset per tutti gli accessi ai database. Un oggetto dataset rappresenta un insieme di record di un certo database organizzati in una tabella logica. Questi record possono essere i record da una singola tabella di database o possono rappresentare i risultati dell'esecuzione di una query o di una procedura registrata.

Tutti gli oggetti dataset che vengono usati nelle applicazioni database discendono da *TDataSet*, ed ereditano da questa classe campi dati, proprietà, eventi e metodi. Questo capitolo descrive le funzionalità di *TDataSet* che vengono ereditate dagli oggetti dataset che si useranno nelle applicazioni database. Per poter utilizzare qualsiasi oggetto dataset è necessario comprendere queste funzionalità condivise.

TDataSet è un dataset 'virtualizzato', nel senso che molte delle sue proprietà e dei suoi metodi sono **virtual** o **pure virtual**. Un *metodo virtual* è una dichiarazione di funzione o di procedura, la cui implementazione può essere (e solitamente lo è) ridefinita negli oggetti discendenti. Un *metodo pure virtual* è una dichiarazione di funzione o di procedura senza una reale implementazione. La dichiarazione è un prototipo che descrive il metodo (e i relativi parametri e tipi restituiti, se ce ne sono) da implementare in tutti gli oggetti dataset discendenti, ognuno dei quali potrebbe però implementarlo in modo differente.

Poiché *TDataSet* contiene metodi **pure virtual**, non può essere usato direttamente in un'applicazione senza generare un errore in esecuzione. Per ovviare a questo inconveniente, è possibile o creare istanze dei discendenti interni di *TDataSet* e usarle nell'applicazione, oppure derivare un proprio oggetto dataset da *TDataSet* o dai suoi discendenti, e scrivere le implementazioni per tutti i suoi metodi **pure virtual**.

TDataSet definisce molte caratteristiche comuni a tutti gli oggetti dataset. *TDataSet* definisce, ad esempio, la struttura di base di tutti i dataset: un array di componenti *TField* che corrispondono alle colonne effettive in una o più tabelle database, campi di consultazione o campi calcolati forniti dall'applicazione. Per maggiori informazioni sui componenti, consultare Capitolo 23, "Operazioni con i componenti campo".

Questo capitolo descrive come utilizzare le comuni funzionalità di database introdotte da *TDataSet*. Tenere presente, comunque, che benché *TDataSet* introduca i metodi per queste funzionalità, non tutti i discendenti da *TDataSet* li implementano. In particolare, i dataset unidirezionali implementano solo un sottoinsieme limitato.

Utilizzo dei discendenti di TDataSet

TDataSet ha numerosi discendenti diretti, ognuno di cui corrisponde a un meccanismo di accesso dati diverso. Non si opera direttamente con nessuno di questi discendenti.Invece, ogni discendente introduce le proprietà e i metodi per l'utilizzo di un particolare meccanismo di accesso ai dati. Queste proprietà e questi metodi sono quindi esposti da classi discendenti che sono adattate a vari tipi di server di dati. I discendenti immediati di *TDataSet* includono

- *TBDEDataSet*, che utilizza Borland Database Engine (BDE) per comunicare con il server di database. I discendenti di *TBDEDataSet* che si utilizzano sono *TTable*, *TQuery*, *TStoredProc* e *TNestedTable*. Le funzioni tipiche dei dataset basati su BDE sono descritte nel Capitolo 24, "Uso di Borland Database Engine".
- *TCustomADODataSet*, che utilizza ActiveX Data Objects (ADO) per comunicare con un datastore OLEDB. I discendenti di *TCustomADODataSet* che si utilizzano sono *TADODataSet*, *TADOTable*, TADOQuery e *TADOStoredProc*. Le funzioni tipiche dei dataset basati su ADO sono descritte nel Capitolo 25, "Operazioni con componenti ADO".
- *TCustomSQLDataSet*, che utilizza dbExpress per comunicare con un server di database. I discendenti di *TCustomSQLDataSet* che si utilizzano sono *TSQLDataSet*, *TSQLTable*, *TSQLQuery* e *TSQLStoredProc*. Le funzioni tipiche dei dataset dbExpress sono descritte nel Capitolo 26, "Uso di dataset unidirezionali".
- *TIBCustomDataSet*, che comunica direttamente con un server di database InterBase. I discendenti di *TIBCustomDataSet* che si utilizzano sono *TIBDataSet*, *TIBTable*, *TIBQuery* e *TIBStoredProc*.
- *TCustomClientDataSet*, che rappresenta i dati da un altro componente dataset o i dati da un file dedicato su disco. I discendenti di *TCustomClientDataSet* che si utilizzano sono *TClientDataSet*, che può collegarsi a un dataset (sorgente) esterno e i dataset client che sono specializzati per un particolare meccanismo di accesso ai dati (*TBDEClientDataSet*, *TSQLClientDataSet* e *TIBClientDataSet*), i quali utilizzano un dataset sorgente interno. Le funzioni tipiche dei dataset client sono descritte nel Capitolo 27, "Utilizzo dei dataset client".

Alcuni pro e contro dei diversi meccanismi di accesso ai dati impiegati da questi discendenti di *TDataSet* sono descritti in "Uso dei databases" a pagina 18-1.

Oltre ai dataset nativi, è possibile creare propri discendenti custom di *TDataSet* - ad esempio per fornire dati da un processo diverso rispetto a un server di database, ad esempio da un foglio elettronico. La scrittura di dataset custom offre la flessibilità di gestire i dati utilizzando un metodo qualsiasi a propria scelta, consentendo al contempo di utilizzare i controlli dati della VCL per costruire l'interfaccia utente. Per

ulteriori informazioni sulla creazione di componenti personalizzati, consultare il Capitolo 45, "Introduzione alla creazione di componenti".

Benché ogni discendente di *TDataSet* abbia proprie proprietà e metodi unici, alcune proprietà e alcuni metodi introdotti dalle classi discendenti sono gli stessi di quelli introdotti da altre classi discendenti che utilizzano un altro meccanismo di accesso di dati. Ad esempio, tra i componenti "tabella" (*TTable*, *TADOTable*, *TSQLTable* e *TIBTable*) vi sono similitudini. Per informazioni sulle caratteristiche comuni introdotte dai discendenti di *TDataSet*, vedere "Tipi di dataset" a pagina 22-24.

Determinazione degli stati del dataset

Lo *stato*—o *modalità*—di un dataset determina le operazioni possibili con i relativi dati. Per esempio, quando un dataset è chiuso, il suo stato è *dsInactive*, ovvero nessuna operazione è consentita sui suoi dati. In fase di esecuzione, è possibile esaminare la proprietà a sola lettura *State* di un dataset per determinare lo stato corrente di quest'ultimo. La seguente tabella riepiloga i valori possibili per la proprietà *State* e il relativo significato:

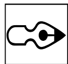
Tabella 22.1 Valori della proprietà State di un dataset

Valore	Stato	Significato
dsInactive	Inactive	Il dataSet è chiuso. I suoi dati non sono disponibili.
dsBrowse	Browse	Il DataSet è aperto. I suoi dati possono essere visualizzati, ma non modificati. Questo è lo stato predefinito di un dataset aperto.
dsEdit	Edit	Il DataSet è aperto. La riga corrente può essere modificata. (non supportato nei dataset unidirezionali)
dsInsert	Insert	Il DataSet è aperto. È possibile inserire una nuova riga. (Non supportato nei dataset unidirezionali)
dsSetKey	SetKey	Il DataSet è aperto. Consente l'impostazione di intervalli e valori chiave usati per gli intervalli e le operazioni *GotoKey*. (Supportato dat tutti i dataset)
dsCalcFields	CalcFields	Il DataSet è aperto. Indica che è in corso un evento *OnCalcFields*. Impedisce di apportare modifiche a campi che non siano calcolati.
dsCurValue	CurValue	Il DataSet è aperto. Indica che la proprietà *CurValue* dei campi viene prelevata da un gestore di evento che risponde agli errori durante l'applicazione degli aggiornamenti in cache.
dsNewValue	NewValue	Il DataSet è aperto. Indica che la proprietà *NewValue* dei campi viene prelevata da un gestore di evento che risponde agli errori durante l'applicazione degli aggiornamenti in cache.
dsOldValue	OldValue	Il DataSet è aperto. Indica che la proprietà *OldValue* dei campi viene prelevata da un gestore di evento che risponde agli errori durante l'applicazione degli aggiornamenti in cache.
dsFilter	Filter	Il DataSet è aperto. Indica che è in corso un'operazione di filtro. È possibile visualizzare una serie limitata di dati, ma non è possibile modificare alcun dato. (Non supportato nei dataset unidirezionali)

Tabella 22.1 Valori della proprietà State di un dataset

Valore	Stato	Significato
dsBlockRead	Block Read	Il DataSet è aperto. I controlli data-aware non vengono aggiornati e gli eventi non vengono innescati quando si modifica il record corrente.
dsInternalCalc	Internal Calc	Il DataSet è aperto. È in corso un evento *OnCalcFields* per i valori calcolati che sono memorizzati con il record. (Solo con dataset client)
dsOpening	Opening	Il dataSet è in fase di apertura ma l'operazione non è terminata. Questo stato si verifica quando il dataset viene aperto per il prelievo asincrono.

Di solito, un'applicazione controlla lo stato di dataset per determinare quando eseguire determinate operazioni. Ad esempio, si potrebbe controllare lo stato *dsEdit* o *dsInsert* per verificare se è necessario registrare gli aggiornamenti.



A ogni cambiamento dello stato di un dataset, l'evento *OnStateChange* viene chiamato per qualsiasi componente datasource associato al dataset. Per ulteriori informazioni sui componenti datasource e su *OnStateChange*, vedere "Risposta alle modifiche mediate dal datasource" a pagina 19-4.

Apertura e chiusura di dataset

Per leggere o modificare i dati in un dataset, un'applicazione deve innanzi tutto aprirlo. Per aprire un dataset è possibile procedere in due modi,

- Impostare la proprietà *Active* del dataset a **true**, in fase di impostazione nell'Object Inspector, oppure in fase di esecuzione nel codice:

```
CustTable->Active = true;
```

- Chiamare il metodo *Open* del dataset in fase di esecuzione,

```
CustQuery->Open();
```

Quando si apre il dataset, il dataset riceve dapprima un evento *BeforeOpen* quindi apre un cursor, si riempie di dati e, infine, riceve un evento *AfterOpen*.

Il dataset appena aperto è in modalità browse, il che significa che l'applicazione può leggere i dati e spostarsi tra di loro.

È possibile chiudere un dataset in due modi,

- Impostare la proprietà *Active* del dataset a **false**, in fase di impostazione nell'Object Inspector, oppure in fase di esecuzione nel codice,

```
CustQuery->Active = false;
```

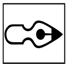
- Chiamare il metodo *Close* per il dataset in fase di esecuzione,

```
CustTable->Close();
```

Esattamente come riceve eventi *BeforeOpen* e *AfterOpen* quando lo si apre, il dataset riceve un evento *BeforeClose* e *AfterClose* quando lo si chiude. È possibile scrivere

gestori di evento che rispondono al metodo *Close* di un dataset. È possibile usare questi eventi, ad esempio, per chiedere a un utente di registrare o annullare le modifiche in sospeso prima di chiudere il dataset. Il seguente codice illustra tale gestore:

```
void __fastcall TForm1::VerifyBeforeClose(TDataSet *DataSet)
{
  if (DataSet->State == dsEdit || DataSet->State == dsInsert)
  {
    TMsgDlgButtons btns;
    btns << mbYes << mbNo;
    if (MessageDlg("Post changes before closing?", mtConfirmation, btns, 0) == mrYes)
      DataSet->Post();
    else
      DataSet->Cancel();
  }
}
```



Quando si desidera modificare determinate proprietà, ad esempio *TableName* di un componente *TTable*, può essere necessario chiudere il dataset. Quando si riapre il dataset, i nuovi valori delle proprietà diventano effettivi.

Spostamenti nei dataset

Ogni dataset attivo ha un *cursore*, o puntatore, che indica la riga corrente del dataset. La *riga corrente* di un dataset è quella i cui valori di campo appaiono nei controlli associati ai dati e riferiti a un singolo campo di una scheda, quali *TDBEdit*, *TDBLabel* e *TDBMemo*. Se il dataset supporta l'editing, il record corrente contiene valori che possono essere trattati dai metodi per l'editing, l'inserimento e la cancellazione.

La riga corrente può essere cambiata spostando il cursore in modo tale che punti a un'altra riga. La seguente tabella elenca i metodi utilizzabili nel codice di un'applicazione per passare ad altri record:

Tabella 22.2 Metodi di navigazione dei dataset

Metodo	Sposta il cursore a
First	La prima riga in un dataset.
Last	L'ultima riga in un dataset. (Non disponibile per dataset unidirezionali)
Next	La riga successiva in un dataset.
Prior	La riga precedente in un dataset. (Non disponibile per dataset unidirezionali)
MoveBy	Un numero di righe specificate in avanti o a ritroso in un dataset.

Il componente visuale associato ai dati *TDBNavigator* incapsula questi metodi come pulsanti selezionabili dagli utenti per spostarsi tra i record in fase di esecuzione. Per ulteriori informazioni sul componente Navigator, vedere "Spostamento e gestione dei record" a pagina 19-31.

Ogni volta che si modifica il record corrente utilizzando uno di questi metodi (o da altri metodi di navigazione che agiscono in base a un criterio di ricerca), il dataset

riceve due eventi: *BeforeScroll* (prima di lasciare il record corrente) e *AfterScroll* (dopo essere arrivato sul nuovo record). È possibile utilizzare questi eventi per aggiornare l'interfaccia utente (ad esempio, per aggiornare una barra di stato riporta informazioni sul record corrente).

TDataSet definisce due proprietà booleane che forniscono informazioni utili sulla posizione del cursore tra i record di un dataset.

Tabella 22.3 Proprietà di navigazione dei dataset

Proprietà	Descrizione
Bof (Beginning-of-file)	**true**: il cursore si trova in corrispondenza della prima riga del dataset.
	false: il cursore non è stato trovato in corrispondenza della prima riga del dataset
Eof (End-of-file)	**true**: il cursore si trova in corrispondenza dell'ultima riga del dataset.
	false: il cursore non è stato trovato in corrispondenza della prima riga del dataset

Utilizzo dei metodi First e Last

Il metodo *First* sposta il cursore alla prima riga di un dataset e imposta la proprietà *Bof* a **true**. Se il cursore è già alla prima riga del dataset, *First* non esegue alcuna operazione.

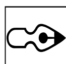
Ad esempio, il seguente codice determina lo spostamento del cursore al primo record di *CustTable*:

```
CustTable->First();
```

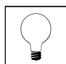
Il metodo *Last* sposta il cursore all'ultima riga di un dataset e imposta la proprietà *Eof* a **true**. Se il cursore è già all'ultima riga del dataset, *Last* non esegue alcuna operazione.

Il seguente codice determina lo spostamento del cursore all'ultimo record di *CustTable*:

```
CustTable->Last();
```



Il metodo *Last* solleva un'eccezione con i dataset unidirezionali.



Anche se possono esservi motivi pratici di programmazione per spostare il cursore alla prima o all'ultima riga di un dataset senza l'intervento dell'utente, lo sviluppatore può anche permettere agli utenti di spostarsi tra i record, avvalendosi del componente *TDBNavigator*. Il componente Navigator contiene una serie di pulsanti che, quando sono attivi e visibili, consentono all'utente di posizionarsi alla prima o all'ultima riga di un dataset attivo. Gli eventi *OnClick* per questi pulsanti chiamano i metodi *First* e *Last* del dataset. Per ulteriori informazioni sull'utilizzo efficace del componente Navigator, vedere "Spostamento e gestione dei record" a pagina 19-31.

Utilizzo dei metodi Next e Prior

Il metodo *Next* sposta il cursore alla riga successiva nel dataset e imposta la proprietà *Bof* a **false** se il dataset non è vuoto. Se il cursore si trova già all'ultima riga del dataset quando si chiama *Next*, non accade nulla.

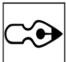
Ad esempio, il seguente codice sposta il cursore al record successivo in *CustTable*:

```
CustTable->Next();
```

Il metodo *Prior* fa arretrare il cursore alla riga precedente nel dataset e imposta *Eof* a **false** se il dataset non è vuoto. Se il cursore si trova già alla prima riga del dataset quando si chiama *Prior*, non accade nulla.

Ad esempio, il seguente codice sposta il cursore al record precedente in *CustTable*:

```
CustTable->Prior();
```

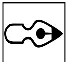


Il metodo *Prior* solleva un'eccezione con i dataset unidirezionali.

Utilizzo del metodo MoveBy

MoveBy consente di specificare quante righe avanti o indietro il cursore deve essere spostato in un dataset. Lo spostamento è relativo al record corrente al momento della chiamata di *MoveBy*. *MoveBy* imposta inoltre in modo appropriato le proprietà *Bof* e *Eof* per il dataset.

Questa funzione legge un parametro rappresentato da un numero intero che indica il numero di record dello spostamento. I numeri interi positivi designano uno spostamento in avanti, mentre i numeri interi negativi designano uno spostamento all'indietro.

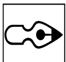


MoveBy solleva un'eccezione con i dataset unidirezionali se si utilizza un argomento con segno negativo.

MoveBy restituisce il numero di righe dello spostamento. Se si cerca di oltrepassare l'inizio o la fine del dataset, il numero di righe restituito da *MoveBy* differisce dal numero di righe richieste per lo spostamento. Questo perché *MoveBy* si ferma quando raggiunge il primo o l'ultimo record del dataset.

Il seguente codice determina lo spostamento a ritroso di due righe in *CustTable*:

```
CustTable->MoveBy(-2);
```



Se l'applicazione usa *MoveBy* in un ambiente database multiutente, occorre tenere presente che i dataset sono dinamici. Se più utenti possono accedere simultaneamente al database e modificarne i dati, un record che un attimo prima era in posizione arretrata di cinque record, nell'arco di pochi istanti, potrebbe essere arretrato di quattro o sei o addirittura di un numero indeterminato di record.

Utilizzo delle proprietà Eof e Bof

Due proprietà a sola lettura utilizzabili in fase di esecuzione, *Eof* (End-of-file) e *Bof* (Beginning-of-file), risultano utili quando si desidera elaborare in modo iterativo tutti i record di un dataset.

Eof

Quando *Eof* è **true**, indica che il cursore è inequivocabilmente posizionato sull'ultima riga di un dataset. *Eof* è impostato a **true** quando un'applicazione

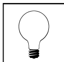
- Apre un dataset vuoto.
- Chiama il metodo *Last* di un dataset.
- Chiama il metodo *Next* di un dataset e il metodo non riesce a compiere alcuna operazione (perché il cursore si trova all'ultima riga del dataset).
- Chiama *SetRange* su un intervallo o dataset vuoto.

In tutti gli altri casi, *Eof* è **false**; si deve presupporre che *Eof* sia **false**, a meno che si verifichi una delle suddette condizioni *e* la proprietà è stata verificata direttamente.

Eof viene comunemente verificata in una condizione di ciclo, per controllare l'elaborazione in sequenza di tutti i record di un dataset. Se viene aperto un dataset contenente record (o se viene chiamato First) *Eof* è **false**. Per elaborare tutti record nel dataset uno per volta, creare un ciclo che si sposti di record in record chiamando il metodo *Next* e che termini quando *Eof* è impostato a **true**. *Eof* rimarrà **false** fino al momento in cui *Next* verrà chiamato quando il cursore è già sull'ultimo record.

Il seguente codice illustra un modo per impostare un ciclo di elaborazione dei record per un dataset denominato *CustTable*:

```
CustTable->DisableControls();
try
{
  for (CustTable->First(); !CustTable->Eof; CustTable->Next())
  (
    // Process each record here
    ⋮
  }
}
__finally
{
  CustTable->EnableControls();
}
```



Questo esempio dimostra anche come disabilitare e abilitare i controlli visuali associati ai dati correlati a un dataset. Disabilitando i controlli visuali durante l'iterazione, l'elaborazione risulta più rapida, in quanto l'applicazione non ha bisogno di aggiornare il contenuto dei controlli quando il record corrente cambia. Al completamento dell'iterazione, è opportuno abilitare nuovamente i controlli per aggiornarli con i valori della nuova riga corrente. Si tenga presente che l'attivazione dei controlli visuali avviene nella clausola **__finally** di un'istruzione **try...__finally**.

Questo assicura che, anche se un'eccezione pone fine prematuramente all'elaborazione del ciclo, i controlli non rimangano disabilitati.

Bof

Quando *Bof* è **true**, indica che il cursore è posizionato inequivocabilmente sulla prima riga di un dataset. *Bof* è **true** quando un'applicazione

- Apre un dataset.
- Chiama il metodo *First* di un dataset.
- Chiama il metodo *Prior* di un dataset e il metodo non riesce a compiere alcuna operazione (perché il cursore si trova alla prima riga del dataset).
- Chiama *SetRange* su un intervallo o dataset vuoto.

Bof è **false** in tutti gli altri casi; si deve presupporre che *Bof* sia **false**, a meno che si verifichi una delle suddette condizioni e la proprietà è stata verificata direttamente.

Come *Eof*, *Bof* può essere inserita in una condizione di ciclo per controllare l'elaborazione in sequenza dei record di un dataset. Il seguente codice illustra un modo per impostare un ciclo di elaborazione dei record per un dataset denominato *CustTable*:

```
CustTable->DisableControls(); // Speed up processing; prevent screen flicker
try
{
  while (!CustTable->Bof) // Cycle until Bof is true
  (
    // Process each record here
    ⋮
    CustTable->Prior();
    // Bof false on success; Bof true when Prior fails on first record
  }
}
__finally
{
  CustTable->EnableControls();
}
```

Contrassegno e ritorno ai record

Oltre che spostarsi da un record al successivo (o impostare un numero specifico di record per lo spostamento), in un dataset è sovente utile contrassegnare un punto specifico di un dataset, per potervi accedere direttamente quando si desidera. TDataSet introduce una funzione di bookmark composta da una proprietà Bookmark e da cinque metodi di bookmark.

TDataSet implementa metodi segnalibro **virtual**. Anche se questi metodi assicurano che qualsiasi oggetto dataset derivato da *TDataSet* restituisca un valore se viene chiamato un metodo segnalibro, i valori di ritorno sono semplici valori predefiniti che non tengono traccia della posizione attuale. I discendenti di *TDataSet* si differenziano per il livello di supporto fornito per i segnalibri. Nessuno dei dataset

dbExpress offre un supporto per i segnalibri. I dataset ADO possono supportare i segnalibri, a seconda delle tabelle di database sottostanti. I dataset BDE, dataset InterBase express e i dataset client supportano sempre i segnalibri.

La proprietà Bookmark

La proprietà *Bookmark* indica quale segnalibro tra un qualsiasi numero di segnalibri dell'applicazione è quello corrente. *Bookmark* è una stringa che identifica il segnalibro attuale.Ogni volta che si aggiunge un altro segnalibro, questo diviene il segnalibro corrente.

Il metodo GetBookmark

Per creare un segnalibro, è necessario dichiarare nell'applicazione una variabile di tipo *TBookmark* e chiamare quindi *GetBookmark* per allocare un'area di memoria per la variabile e impostarne il valore a una posizione specifica del dataset. La variabile di tipo *TBookmark* è un puntatore (void *).

I metodi GotoBookmark e BookmarkValid

Quando riceve un segnalibro, *GotoBookmark* sposta il cursore del dataset alla posizione specificata nel segnalibro. Prima di chiamare *GotoBookmark* si può chiamare *BookmarkValid* per determinare se il segnalibro punta a un record. *BookmarkValid* restituisce **true** se il segnalibro specificato punta a un record.

Il metodo CompareBookmarks

Si può chiamare *CompareBookmarks* anche per vedere se un segnalibro al quale ci si desidera muovere è diverso da un altro (o da quello corrente). Se i due segnalibri fanno riferimento allo stesso record (o se entrambi sono NULL), *CompareBookmarks* restituisce 0.

Il metodo FreeBookmark

FreeBookmark libera la memoria assegnata a un determinato segnalibro, quando quest'ultimo non serve più. È opportuno chiamare *FreeBookmark* prima di riutilizzare un segnalibro esistente.

Un esempio di utilizzo dei segnalibri

Il seguente codice illustra un impiego dei segnalibri:

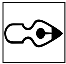
```
void DoSomething (const TTable *Tbl)
{
  TBookmark Bookmark = Tbl->GetBookmark(); // Allocate memory and assign a value
  Tbl->DisableControls(); // Turn off display of records in data controls
  try
  {
    for (Tbl->First(); !Tbl->Eof; Tbl->Next()) // Iterate through each record in table
    {
      // Do your processing here
      ⋮
    }
  }
```

```
    __finally
    {
      Tbl->GotoBookmark(Bookmark);
      Tbl->EnableControls(); // Turn on display of records in data controls
      Tbl->FreeBookmark(Bookmark); // Deallocate memory for the bookmark
    }
  }
```

Prima di passare in sequenza tutti i record, i controlli vengono disabilitati. Qualora si verifichi un errore durante l’iterazione tra i record, la clausola **__finally** garantisce che i controlli vengano sempre abilitati e che il segnalibro sia sempre liberato anche se il ciclo termina prematuramente.

Ricerche nei dataset

Se un dataset non è unidirezionale, è possibile effettuare ricerche su di esso utilizzando i metodi *Locate* e *Lookup*. Questi metodi permettono di ricercare su qualsiasi tipo di colonne in qualsiasi dataset.



TDataSet introducono una famiglia aggiuntiva di metodi per effettuare ricerche in base a un indice. Per informazioni su questi metodi aggiuntivi, vedere “Utilizzo di indici per la ricerca di record” a pagina 22-29.

Uso di Locate

Locate posiziona il cursore sulla prima riga corrispondente a un insieme specificato di criteri di ricerca. Nella forma più semplice, a *Locate* viene passato il nome di una colonna in cui eseguire la ricerca, un valore di campo che rappresenta il criterio di ricerca e un flag di opzioni che specifica se la ricerca deve differenziare i caratteri maiuscoli e minuscoli o se è valida la corrispondenza parziale alle chiavi. (La corrispondenza parziale delle chiavi si verifica quando la stringa di criterio è uguale al valore iniziale del valore del campo). Ad esempio, il seguente codice sposta il cursore alla prima riga di *CustTable*, dove il valore della colonna *Company* è “Professional Divers, Ltd.”:

```
TLocateOptions SearchOptions;
SearchOptions.Clear();
SearchOptions << loPartialKey;
bool LocateSuccess = CustTable->Locate(“Company”, “Professional Divers, Ltd.”,
    SearchOptions);
```

Se *Locate* trova una corrispondenza, il primo record contenente la corrispondenza diventa il record corrente. *Locate* restituisce **true** se trova un record corrispondente, **false** se non ne trova. Se non vengono trovate corrispondenze, il record corrente non cambia.

La vera potenza di *Locate* può essere apprezzata quando si desidera effettuare ricerche su più colonne e specificare valori multipli da cercare. I valori da cercare sono Variant, pertanto è possibile specificare tipi di dati diversi nei criteri di ricerca. Per specificare più colonne in una stringa di ricerca, separare le singole voci della stringa con un punto e virgola.

Poiché i valori da cercare sono Variant, se si passano più valori è necessario o passare come argomento un array Variant (per esempio, i valori restituiti dal metodo Lookup), oppure occorre costruire nel codice l'array Variant utilizzando la funzione *VarArrayOf*. Il seguente codice illustra una ricerca eseguita su più colonne, adottando valori di ricerca multipli e accettando la corrispondenza parziale alle chiavi:

```
TLocateOptions Opts;
Opts.Clear();
Opts << loPartialKey;
Variant locvalues[2];
locvalues[0] = Variant("Sight Diver");
locvalues[1] = Variant("P");
CustTable->Locate("Company;Contact", VarArrayOf(locvalues, 1), Opts);
```

Locate adotta il metodo più veloce possibile per trovare i record corrispondenti. Se le colonne in cui effettuare la ricerca sono indicizzate e l'indice è compatibile con le opzioni di ricerca specificate, *Locate* utilizza l'indice.

Uso di Lookup

Lookup cerca la prima riga corrispondente ai criteri di ricerca specificati. Se trova una riga corrispondente, determina la rielaborazione dei campi calcolati e dei campi di ricerca associati al dataset, quindi restituisce uno o più campi tratti dalla riga corrispondente. *Lookup* non sposta il cursore sulla riga corrispondente, ma restituisce solo valori estratti da essa.

Nella forma più semplice, a *Lookup* viene passato il nome del campo in cui eseguire la ricerca, il valore di campo che rappresenta il criterio di ricerca e il campo o i campi da restituire. Per esempio, il seguente codice cerca il primo record di *CustTable* in cui il valore del campo *Company* sia "Professional Divers, Ltd." e restituisce il nome dell'azienda, la persona da contattare e il numero di telefono dell'azienda:

```
Variant LookupResults = CustTable->Lookup("Company", "Professional Divers, Ltd",
    "Company;Contact;Phone");
```

Lookup restituisce i valori per i campi specificati dal primo record corrispondente che trova. I valori vengono restituiti come Variant. Se viene richiesto più di un valore, *Lookup* restituisce un array Variant. Se non esiste alcun record corrispondente, *Lookup* restituisce un Variant Null. Per maggiori informazioni sugli array Variant, consultare la Guida in linea.

La vera potenza di *Lookup* entra in gioco quando si vogliono effettuare ricerche su più colonne e si specificano più valori da cercare. Per specificare stringhe contenenti più colonne o campi risultato, separare i singoli campi degli elementi della stringa con dei punti e virgola.

Poiché i valori da cercare sono Variant, se si passano più valori è necessario o passare come argomento un array Variant (per esempio, i valori restituiti dal metodo *Lookup*), oppure occorre costruire nel codice l'array Variant utilizzando la funzione *VarArrayOf*. Il codice seguente illustra una ricerca di lookup su più colonne:

```
Variant LookupResults;
Variant locvalues[2];
Variant v;
```

```
locvalues[0] = Variant("Sight Diver");
locvalues[1] = Variant("Kato Paphos");
LookupResults = CustTable->Lookup("Company;City", VarArrayOf(locvalues, 1),
                  "Company;Addr1;Addr2;State;Zip");
// now put the results in a global stringlist (created elsewhere)
pFieldValues->Clear();
for (int i = 0; i < 5; i++) // Lookup call requested 5 fields
{
  v = LookupResults.GetElement(i);
  if (v.IsNull())
    pFieldValues->Add("");
  else
    pFieldValues->Add(v);
}
```

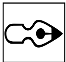
Come *Locate*, *Lookup* adotta il metodo più veloce possibile per trovare i record corrispondenti. Se le colonne in cui effettuare la ricerca sono indicizzate, *Lookup* utilizza l'indice.

Visualizzazione e modifica di un sottoinsieme di dati usando i filtri

Spesso un'applicazione è interessata solo a un sottoinsieme di record di un dataset. Per esempio, può essere necessario reperire o visualizzare solo i record relativi alle aziende con sede in un determinato Paese nel database dei clienti, oppure può servire solo un record contenente un insieme specifico di valori di campi. In ciascun caso, è possibile usare filtri per restringere l'accesso di un'applicazione a un sottoinsieme di tutti i record del dataset.

Con i dataset unidirezionali, è possibile ridurre i record nel dataset solo se si specifica una query che limita il numero di record nel dataset. Con altri discendenti di *TDataSet*, tuttavia, è possibile definire un sottoinsieme dei dati che sono già stati prelevati. Per limitare l'accesso di un'applicazione a un sottoinsieme di tutti i record nel dataset, è possibile utilizzare filtri.

Un filtro specifica le condizioni che un record deve soddisfare per essere visualizzato. Le condizioni del filtro possono essere stipulate nella proprietà *Filter* di un dataset o possono essere codificate nel suo gestore di evento *OnFilterRecord*. Le condizioni del filtro sono basate sui valori di un qualunque numero di campi di un dataset, siano essi indicizzati o meno. Ad esempio, per visualizzare solamente i record relativi alle aziende con sede in California, un semplice filtro richiederebbe che i record contengano il valore "CA" nel campo State.



I filtri valgono per ogni record reperito in un dataset. Quando devono essere filtrati volumi ingenti di dati, invece di utilizzare i filtri può essere più efficace ricorrere a una query per limitare il reperimento dei record, oppure impostare un intervallo in una tabella indicizzata.

Attivazione e disattivazione dei filtri

L'attivazione dei filtri su un dataset è un processo in tre fasi:

1 Creazione di un filtro.
2 Impostazione, se occorre, delle opzioni per i filtri basati su stringhe.
3 Impostazione della proprietà *Filtered* a **true**.

Quando il filtro è attivo, solamente i record che ne soddisfano i criteri sono disponibili a un'applicazione. Il filtro è sempre una condizione provvisoria. Il filtro può essere disattivato impostando la proprietà *Filtered* a **false**.

Creazione di filtri

Ci sono due modi per creare un filtro per un dataset:

- Specificare semplici condizioni di filtro nella proprietà *Filter*. *Filter* è particolarmente utile per creare e applicare filtri in fase di esecuzione.
- Scrivere un gestore di evento *OnFilterRecord* per condizioni del filtro semplici o complesse. Con *OnFilterRecord*, si specificano in fase di progetto le condizioni di filtro. A differenza della proprietà *Filter*, che è limitata a una singola stringa che contiene la logica del filtro, un evento *OnFilterRecord* può sfruttare salti condizionali e cicli per creare complesse condizioni di filtro multi-livello.

Il vantaggio principale di creare filtri usando la proprietà *Filter* è che un'applicazione può creare, modificare e applicare dinamicamente filtri, (per esempio, in risposta a un input dell'utente). Lo svantaggio principale consiste nel fatto che le condizioni del filtro devono essere esprimibili in una singola stringa di testo, non si può far uso di costrutti di diramazione condizionale o ciclici, e non è possibile verificare o confrontare valori con valori che non siano già presenti nel dataset.

I vantaggi dell'evento *OnFilterRecord* sono che un filtro può essere complesso e variabile, può essere basato su molte righe di codice che usano strutture condizionali e cicliche, e può confrontare valori del dataset con valori esterni al dataset, come ad esempio il testo in una casella di testo. Il principale punto debole di *OnFilterRecord* è che si imposta il filtro in fase di progetto e quindi non può essere cambiato in risposta a un input dell'utente. (È comunque possibile creare molti gestori di filtro e passare dall'uno all'altro in risposta alle condizioni generali di un'applicazione).

Le sezioni che seguono descrivono come creare filtri usando la proprietà *Filter* e il gestore di evento *OnFilterRecord*.

Impostazione della proprietà Filter

Per creare un filtro usando la proprietà *Filter*, impostare il valore della proprietà a una stringa che contiene le condizioni del filtro. L'istruzione che segue, ad esempio, crea un filtro che verifica se il campo State di un dataset contiene un valore per lo stato della California:

```
Dataset1->Filter = "State = 'CA'";
```

È anche possibile fornire un valore a *Filter* in base al testo immesso in un controllo. L'istruzione che segue, ad esempio, assegna a *Filter* il testo di una casella di testo:

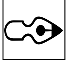
```
Dataset1->Filter = Edit1->Text;
```

Ovviamente, è possibile creare una stringa basata su un testo inserito a livello di codice e sui dati immessi dall'utente:

```
Dataset1->Filter = AnsiString("State = '") + Edit1->Text + "'";
```

I valori dei campi vuoti non vengono visualizzati a meno che non siano inclusi esplicitamente nel filtro:

```
Dataset1->Filter = "State <> 'CA' or State = BLANK";
```



Dopo aver specificato un valore di *Filter*, si imposti a **true** la proprietà *Filtered* per applicare il filtro al dataset.

I filtri possono confrontare i valori dei campi con letterali e costanti usando i seguenti operatori logici e di confronto:

Tabella 22.4 Operatori logici e di confronto che possono apparire in un filtro

Operatore	Significato
<	Minore di
>	Maggiore di
>=	Maggiore o uguale a
<=	Minore o uguale a
=	Uguale a
<>	Non uguale a
AND	Verifica che due istruzioni siano entrambe **true**
NOT	Verifica che l'istruzione seguente non sia **true**
OR	Verifica che almeno uno delle due istruzioni sia **true**
+	Aggiunge numeri, concatena stringhe, aggiunge numeri a valori data/ora (disponibile solo per alcuni driver)
-	Sottrae numeri, sottrae date o sottrae un numero da una data (disponibile solo per alcuni driver)
*	Moltiplica due numeri (disponibile solo per alcuni driver)
/	Divide due numeri (disponibile solo per alcuni driver)
*	carattere jolly per confronti parziali (*FilterOptions* deve includere *foPartialCompare*)

Usando combinazioni di questi operatori, è possibile creare filtri abbastanza sofisticati. L'istruzione che segue, ad esempio, si assicura che le due condizioni di verifica siano soddisfatte prima di accettare un record da visualizzare:

```
(Custno > 1400) AND (Custno < 1500);
```



Quando il filtro è attivato, le modifiche apportate da un utente a un record possono determinare che quest'ultimo non corrisponda più alle condizioni verificate dal filtro. Al successivo reperimento del record nel dataset, il record può dunque "scomparire". In questo caso, il record successivo che passa la condizione di filtro diventa il record corrente.

Scrittura di un gestore di eventi OnFilterRecord

Per indicare se un record soddisfa la condizione di filtro, il gestore di evento *OnFilterRecord* imposta il parametro *Accept* a **true** per includere un record o a **false** per escluderlo. Per esempio, il seguente filtro visualizza solo i record con il campo State impostato su ”CA”.

```
void __fastcall TForm1::Table1FilterRecord(TDataSet *DataSet; bool &Accept)
{
  Accept = DataSet->FieldByName[”State”]->AsString == ”CA”;
}
```

Quando il filtro è attivo, per ciascun record reperito viene generato un evento *OnFilterRecord*. Il gestore di evento verifica ciascun record e visualizza solamente quelli che soddisfano le condizioni del filtro. Poiché l'evento *OnFilterRecord* viene generato per ogni record di un dataset, il codice del gestore di evento dovrebbe essere il più breve possibile per evitare che influenzi negativamente le prestazioni dell'applicazione.

Commutazione dei gestori evento di filtro in esecuzione

È possibile programmare un numero qualsiasi di gestori evento *OnFilterRecord* e passare dall'uno all'altro in fase di esecuzione. Le istruzioni che seguono, ad esempio, passano a un gestore di evento *OnFilterRecord*, chiamato *NewYorkFilter*:

```
DataSet1->OnFilterRecord = NewYorkFilter;
Refresh();
```

Impostazione delle opzioni del filtro

La proprietà *FilterOptions* consente di specificare se un filtro che confronta campi basati su stringhe accetta i record in base a un confronto parziale e se il confronto delle stringhe deve tener conto delle maiuscole/minuscole. *FilterOptions* è una proprietà set che può essere un set vuoto (impostazione predefinita), o che può contenere l'uno o l'altro o entrambi i seguenti valori:

Tabella 22.5 Valori di FilterOptions

Valore	Significato
foCaseInsensitive	Ignora maiuscolo/minuscolo durante il confronto delle stringhe.
foNoPartialCompare	Disattiva la corrispondenza parziale di stringa; vale a dire che le stringhe che terminano con un asterisco (*) non vengono confrontate.

Le istruzioni che seguono, ad esempio, prepararono un filtro che, durante il confronto dei valori nel campo *State*, non tiene conto delle maiuscole/minuscole:

```
TFilterOptions FilterOptions;
FilterOptions->Clear();
FilterOptions << foCaseInsensitive;
Table1->FilterOptions = FilterOptions;
Table1->Filter = ”State = 'CA'”;
```

Spostamento tra i record in un dataset filtrato

Esistono quattro metodi del dataset che consentono di spostarsi tra i record di un dataset filtrato. La tabella che segue elenca questi metodi e descrive ciò che fanno:

Tabella 22.6 Metodi di navigazione nei dataset filtrati

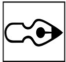
Metodo	Scopo
FindFirst	Si sposta al primo record che corrisponde al criterio di filtro corrente. La ricerca del primo record corrispondente comincia sempre dal primo record del dataset non filtrato.
FindLast	Si muove all'ultimo record che corrisponde al criterio di filtro corrente.
FindNext	Si sposta dal record attuale del dataset filtrato al successivo.
FindPrior	Si sposta dal record corrente del dataset filtrato a quello precedente.

L'istruzione che segue, ad esempio, trova il primo record filtrato di un dataset:

```
DataSet1->FindFirst();
```

Purché si imposti la proprietà *Filter* o si crei un gestore di evento *OnFilterRecord* per un'applicazione, questi metodi posizionano il cursore sul record specificato indipendentemente dal fatto che il filtraggio si attivo o meno. Se si chiamano questi metodi quando il filtro non è attivo

- Attivano temporaneamente il filtro.
- Posizionano il cursore su un record corrispondente, se ne viene trovato uno.
- Disattivano il filtro.



Se il filtro è disattivato e non si imposta la proprietà *Filter* o si crea un gestore di evento *OnFilterRecord*, questi metodi non fanno alcunché di diverso da *First()*, *Last()*, *Next()* e *Prior()*.

Tutti i metodi filtro di spostamento posizionano il cursore su un record corrispondente, se ne è stato trovato uno, rendendolo il record attivo, e restituiscono **true**. Se non si trova un record corrispondente, la posizione del cursore rimane immutata e questi metodi restituiscono **false**. È possibile controllare lo stato della proprietà *Found* per inglobare queste chiamate, ed eseguire un'azione solo quando *Found* è **true**. Se, ad esempio, il cursore è già sull'ultimo record corrispondente del dataset e si chiama *FindNext*, il metodo restituisce **false** e il record corrente non cambia.

Modifica dei dati

È possibile utilizzare i metodi del dataset sotto elencati per inserire, aggiornare e cancellare dati se la proprietà a sola lettura *CanModify* è **true**. *CanModify* è **true** a meno che il dataset non sia unidirezionale, che il database sottostante al dataset non consenta i diritti di lettura e di scrittura o non intervengano altri fattori. (Questi

fattori includono la proprietà *ReadOnly* di alcuni dataset o la proprietà *RequestLive* dei componenti *TQuery*).

Tabella 22.7 Metodi dei dataset per inserire, aggiornare e cancellare dati

Metodo	Descrizione
Edit	Imposta il dataset in modalità *dsEdit*, se già non è nello stato *dsEdit* o *dsInsert*.
Append	Invia i dati in attesa, sposta il record attivo alla fine del dataset e pone il dataset in modalità *dsInsert*.
Insert	Invia i dati in attesa e pone il dataset in modalità *dsInsert*.
Post	Cerca di registrare nel database il record nuovo o modificato. Se l'operazione riesce, il dataset torna allo stato *dsBrowse*; se non riesce, il dataset rimane nello stato corrente.
Cancel	Annulla l'operazione corrente e pone il dataset in modalità *dsBrowse*.
Delete	Cancella il record corrente e pone il dataset in modalità *dsBrowse*.

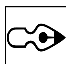
Modifica dei record

Un dataset deve essere in modalità *dsEdit* perché un'applicazione possa modificare i record. Nel codice, il metodo *Edit* può essere utilizzato per impostare un dataset in modalità *dsEdit* se la proprietà a sola lettura *CanModify* del dataset è **true**.

Quando un dataset passa in modalità *dsEdit*, riceve dapprima un evento *BeforeEdit*. Dopo il passaggio alla modalità editing, il dataset riceve un evento *AfterEdit*. Di solito, questi eventi sono utilizzati per l'aggiornamento dell'interfaccia utente per indicare lo stato corrente del dataset. Se il dataset non può essere posto in modalità editing per un qualche motivo, si verifica un evento *OnEditError*, in cui è possibile informare l'utente del problema o provare a correggere la situazione che ha impedito al dataset di passare in modalità editing.

Nelle schede dell'applicazione, alcuni controlli associati ai dati possono portare automaticamente un dataset allo stato *dsEdit* se

- La proprietà *ReadOnly* del controllo è **false** (impostazione predefinita),
- La proprietà *AutoEdit* del *datasource* è **true** e
- *CanModify* è **true** per il dataset.



Anche se un dataset è nello stato *dsEdit*, la modifica dei record può non riuscire per i database basati su *SQL* se l'utente dell'applicazione non possiede i privilegi di accesso SQL appropriati.

Quando un dataset è in modalità *dsEdit*, un utente può modificare i valori dei campi del record corrente che appaiono nei controlli associati ai dati di una scheda. I controlli associati ai dati per i quali la modifica è consentita chiamano automaticamente *Post* quando un utente esegue un'operazione che cambia il record corrente (per esempio lo spostamento a un altro record in una griglia).

Se nelle schede è previsto un componente Navigator, gli utenti possono annullare le modifiche facendo clic sul pulsante *Cancel* del Navigator. Annullando le modifiche, viene ripristinato lo stato *dsBrowse* del dataset.

Nel codice, è necessario scrivere o annullare le modifiche chiamando i metodi appropriati. Le modifiche vengono scritte chiamando *Post*. Possono essere annullate chiamando *Cancel*. Nel codice, *Edit* e *Post* sono sovente utilizzati insieme. Per esempio,

```
Table1->Edit();
Table1->FieldValues["CustNo"] = 1234;
Table1->Post();
```

Nell'esempio precedente, la prima riga di codice imposta la modalità *dsEdit* per il dataset. La successiva riga di codice assegna la stringa "1234" al campo *CustNo* del record corrente. Infine, l'ultima riga scrive, ovvero registra, il record modificato nel database. Se non si utilizzano gli aggiornamenti in cache, la registrazione scrive le modifiche nel database di origine. Se si utilizzano gli aggiornamenti in cache, la modifica vengono scritte in un buffer temporaneo, in cui restano finché non viene chiamato il metodo *ApplyUpdates* del dataset.

Aggiunta di nuovi record

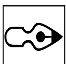
Un dataset deve essere in modalità *dsInsert* perché un'applicazione possa aggiungervi nuovi record. Nel codice, è possibile utilizzare i metodi *Insert* o *Append* per portare un dataset in modalità *dsInsert*, a patto che la proprietà a sola lettura *CanModify* del dataset sia **true**.

Quando un dataset passa in modalità *dsInsert*, riceve dapprima un evento *BeforeInsert*. Dopo che il completamento con successo del passaggio alla modalità inserimento, il dataset riceve prima un evento *OnNewRecord* e quindi un evento *AfterInsert*. È possibile usare questi eventi, ad esempio, per fornire dei valori iniziali per i record appena inseriti:

```
void __fastcall TForm1::OrdersTableNewRecord(TDataSet *DataSet)
{
  DataSet->FieldByName("OrderDate")->AsDateTime = Date();
}
```

Nelle schede dell'applicazione, i controlli griglia data-awar e Navigator possono impostare un dataset in stato *dsInsert* se

- La proprietà *ReadOnly* del controllo è **false** (impostazione predefinita) e
- *CanModify* è **true** per il dataset.



Anche se un dataset è nello stato *dsInsert*, l'aggiunta di record può non andare a buon fine per i database basati su SQL nel caso l'utente dell'applicazione non abbia i privilegi di accesso SQL appropriati.

Quando un dataset è in modalità *dsInsert*, un utente o un'applicazione può immettere valori nei campi associati al nuovo record. Ad eccezione dei controlli griglia e Navigator, non esistono differenze visibili a un utente tra *Insert* e *Append*. Chiamando *Insert*, nella griglia appare una riga vuota sopra quello che era il record corrente. Chiamando *Append*, la griglia viene fatta scorrere fino all'ultimo record del dataset, appare una riga vuota nella parte inferiore della griglia e i pulsanti *Next* e *Last* vengono disattivati in qualsiasi componente Navigator associato al dataset.

I controlli data-aware per i quali è consentito l'inserimento chiamano automaticamente *Post* quando un utente esegue un'operazione che cambia il record corrente (per esempio lo spostamento a un altro record in una griglia). In caso contrario, *Post* deve essere chiamato nel codice.

Post scrive il nuovo record nel database oppure, se sono abilitati gli aggiornamenti in cache, *Post* scrive il record in una cache in memoria. Per scrivere nel database gli inserimenti e le aggiunte in cache, chiamare il metodo *ApplyUpdates* del dataset.

Inserimento di record

Insert apre un nuovo record vuoto prima del record corrente e imposta il record vuoto come record corrente, cosicché l'utente o il codice dell'applicazione possa immettervi valori di campi.

Quando un'applicazione chiama *Post* (o *ApplyUpdates* se gli aggiornamenti in cache sono consentiti), un record appena inserito può essere scritto nel database in tre modi:

- Per le tabelle Paradox e dBASE indicizzate, il record viene inserito nel dataset in una posizione basata sul relativo indice.
- Per le tabelle Paradox e dBASE non indicizzate, il record viene inserito nel dataset nella posizione corrente.
- Per i database SQL, il punto di inserimento fisico dipende dall'implementazione. Se la tabella è indicizzata, l'indice viene aggiornato con le informazioni del nuovo record.

Aggiunta di record

Append apre un nuovo record vuoto alla fine del dataset e lo imposta come record corrente, cosicché un utente o il codice dell'applicazione possa immettervi valori di campi.

Quando un'applicazione chiama *Post* (o *ApplyUpdates* se gli aggiornamenti in cache sono consentiti), un record appena inserito può essere scritto nel database in tre modi:

- Per le tabelle Paradox e dBASE indicizzate, il record viene inserito nel dataset in una posizione basata sul relativo indice.
- Per le tabelle Paradox e dBASE non indicizzate, il record viene aggiunto alla fine del dataset.
- Per i database SQL, il punto fisico in cui avviene l'aggiunta dipende dall'applicazione. Se la tabella è indicizzata, l'indice viene aggiornato con le informazioni del nuovo record.

Cancellazione dei record

Utilizzare il metodo *Delete* per cancellare il record corrente in un dataset attivo. Quando si chiama il metodo *Delete*,

- Il dataset riceve un evento *BeforeDelete*.
- Il dataset tenta di cancellare il record corrente.
- Il dataset ritorna allo stato *dsBrowse*.
- Il dataset riceve un evento*AfterDelete*.

Se si vuole evitare la cancellazione nel gestore di evento *BeforeDelete*, si può chiamare la procedura globale *Abort*:

```
void __fastcall TForm1::TableBeforeDelete (TDataSet *Dataset)
{
  if (MessageBox(0, "Delete This Record?", "CONFIRM", MB_YESNO) != IDYES)
   Abort();
}
```

Se *Delete* non va a buon fine, genera un evento *OnDeleteError*. Se il gestore di evento *OnDeleteError* non può risolvere il problema, il dataset rimane nello stato *dsEdit*. Se *Delete* va a buon fine, il dataset ritorno allo stato *dsBrowse* e il record successivo a quello cancellato diventa il record corrente.

Se si utilizzano gli aggironamenti in cache, il record cancellato non viene rimosso dalla tabella di database sottostante finché non si chiama *ApplyUpdates*.

Se nelle schede è previsto un componente navigator, gli utenti possono cancellare il record corrente facendo clic sul pulsante *Delete* del Navigator. Nel codice, è necessario chiamare esplicitamente *Delete* per rimuovere il record corrente.

Registrazione dei dati

Dopo avere finito di modificare un record, è necessario chiamare il metodo *Post* per scrivere le modifiche. Il metodo *Post* si comporta in modo diverso, a seconda dello stato del dataset e del fatto che si stiano utilizzando gli aggiornamenti in cache.

- Se non si utilizzano gli aggiornamenti in cache e il dataset è nello stato *dsEdit* o *dsInsert*, *Post* scrive il record corrente nel database e riporta il dataset allo stato *dsBrowse*.
- Se si utilizzano gli aggiornamenti in cache e il dataset è nello stato *dsEdit* o *dsInsert*, *Post* scrive il record corrente in una cache interna e riporta il dataset allo stato *dsBrowse*. Le modifiche non vengono scritto nel database finché non si chiama *ApplyUpdates*.
- Se il dataset è in modalità *dsSetKey*, *Post* riporta il dataset allo stato *dsBrowse*.

Indipendentemente dallo stato iniziale del dataset, *Post* genera eventi *BeforePost* e *AfterPost*, prima e dopo aver scritto le modifiche correnti. È possibile utilizzare questi eventi per aggiornare l’interfaccia utente o impedire al dataset registrare le modifiche chiamando la procedura *Abort*. Se la chiamata a *Post* non riesce, il dataset riceve un evento *OnPostError*, dove è possibile informare l’utente del problema o tentare di correggerlo.

La registrazione può essere effettuata esplicitamente o implicitamente nell’ambito di un’altra procedura. Quando un’applicazione si sposta dal record corrente, *Post* viene chiamato implicitamente. Le chiamate ai metodi *First*, *Next*, *Prior* e *Last* eseguono

un'operazione *Post* se la tabella è in modalità *dsEdit* o *dsInsert*. Anche i metodi *Append* e *Insert* registrano implicitamente i dati in sospeso.



Il metodo *Close* non chiama implicitamente *Post*. Per registrare in modo esplicito qualsiasi modifica in sospeso, occorre usare l'evento *BeforeClose*.

Annullamento delle modifiche

Un'applicazione può annullare in qualsiasi momento le modifiche apportate al record corrente, se non ha ancora chiamato direttamente o indirettamente *Post*. Per esempio, se un dataset è in modalità *dsEdit* e un utente modifica i dati di uno o più campi, l'applicazione può ripristinare i valori originali del record chiamando il metodo *Cancel* del dataset. Una chiamata a *Cancel* riporta sempre il dataset allo stato *dsBrowse*.

Se nel momento in cui l'applicazione ha chiamato *Cancel* il dataset era in modo *dsEdit* o *dsInsert*, il dataset riceve gli eventi *BeforeCancel* e *AfterCancel* prima e dopo che il record corrente è stato ripristinato ai suoi valori originali.

Nelle schede, è possibile consentire agli utenti di annullare le operazioni di modifica, inserimento o aggiunta includendo un pulsante *Cancel* in un componente Navigator associato al dataset, oppure è possibile specificare il codice del pulsante *Cancel* nella scheda.

Modifica di interi record

Nelle schede, tutti i controlli associati ai dati, ad eccezione dei controlli griglia e Navigator, forniscono l'accesso a un unico campo di un record.

Tuttavia nel codice è possibile adottare i seguenti metodi che funzionano con intere strutture di record, purché la struttura delle tabelle di database associate al dataset sia stabile e non cambi. La seguente tabella riepiloga i metodi disponibili per lavorare con interi record invece che con i singoli campi interni:

Tabella 22.8 Metodi che funzionano con interi record

Metodo	Descrizione
AppendRecord([array of values])	Aggiunge un record con i valori di colonna specificati alla fine di una tabella; è analogo ad *Append*. Esegue un'operazione *Post* implicita.
InsertRecord([array of values])	Inserisce i valori specificati come record prima della posizione corrente del cursore in una tabella; è analogo a *Insert*. Esegue un'operazione *Post* implicita.
SetFields([array of values])	Imposta i valori dei campi corrispondenti; è analogo all'assegnazione dei valori a *TFields*. L'applicazione deve eseguire un'operazione *Post* esplicita.

Questi metodi accettano come argomento un array di valori, in cui ciascun valore corrisponde a una colonna nel dataset sottostante. Per creare questi array, si deve usare la macro ARRAYOFCONST. I valori possono essere letterali, variabili o NULL.

Se il numero di valori di un argomento è inferiore al numero di colonne di un dataset, i valori rimanenti sono considerati NULL.

Per i dataset non indicizzati, *AppendRecord* aggiunge un record alla fine del dataset, mentre *InsertRecord* inserisce un record dopo la posizione corrente del cursore. Per le tabelle indicizzate, entrambi i metodi inseriscono il record nella tabella alla posizione corretta, in base all'indice. In entrambi i casi, i metodi spostano il cursore alla posizione del record.

SetFields assegna i valori specificati nell'array di parametri ai campi del dataset. Per utilizzare *SetFields*, un'applicazione deve innanzi tutto chiamare *Edit* per portare il dataset in modalità *dsEdit*. Per applicare le modifiche al record corrente, deve eseguire un'operazione *Post*.

Se si usa *SetFields* per modificare alcuni campi di un record esistente, ma non tutti, è possibile passare valori NULL per i campi che non devono essere modificati. Se non vengono forniti abbastanza valori per tutti i campi di un record, *SetFields* assegna ad essi valori NULL. I valori NULL sovrascrivono tutti i valori già esistenti in quei campi.

Ad esempio, si supponga che un database abbia una tabella COUNTRY contenenti le colonne *Name*, Capital, Continent, Area e Population. Se un componente *TTable* denominato *CountryTable* fosse collegato alla tabella COUNTRY, la seguente istruzione inserirebbe un record nella tabella COUNTRY:

```
CountryTable->InsertRecord(ARRAYOFCONST((“Japan”, “Tokyo”, “Asia”)));
```

Questa istruzione non specifica valori per Area e Population, pertanto in corrispondenza di tali campi vengono inseriti valori NULL. La tabella è indicizzata in base a *Name*, quindi l'istruzione inserirebbe il record in base alla collazione alfabetica di “Japan”.

Per aggiornare il record, un'applicazione potrebbe usare il seguente codice:

```
TLocateOptions SearchOptions;
SearchOptions->Clear();
SearchOptions << loCaseInsensitive;
if (CountryTable->Locate(“Name”, “Japan”, SearchOptions))
{
  CountryTable->Edit();
  CountryTable->SetFields(ARRAYOFCONST(((void *)NULL, (void *)NULL, (void *)NULL,
        344567, 164700000)));
  CountryTable->Post();
}
```

Questo codice assegna i valori ai campi Area e Population e quindi li registra nel database. I tre puntatori NULL si comportano come segnaposto per le prime tre colonne, in modo da preservarne il contenuto attuale.



Quando si usano puntatori NULL con *SetFields* per lasciare inalterati alcuni valori di campo, bisogna accertarsi di effettuare la conversione da NULL a puntatore void *. Se si usa NULL come parametro senza effettuare la conversione di tipo, il campo verrà impostato ad un valore vuoto.

Calcolo dei campi

Utilizzando *Fields* editor, è possibile definire dei campi calcolati per i dataset. Quando un dataset contiene campi calcolati, si deve preparare del codice per calcolare quei valori di campo in un gestore di evento *OnCalcFields*. Per ulteriori informazioni sulla definizione dei campi calcolati con l'editor *Fields*, consultare "Definizione di un campo calcolato" a pagina 23-7.

La proprietà *AutoCalcFields* determina quando *OnCalcFields* viene chiamato. Se *AutoCalcFields* è **true**, *OnCalcFields* viene chiamato quando

- Un dataset viene aperto.
- Il dataset passa in modalità editing.
- Un record viene reperito nel database.
- Il fuoco si sposta da un componente visivo a un altro o da una colonna a un'altra in un controllo di griglia associata ai dati e il record attivo è stato modificato.

Se *AutoCalcFields* è **false**, *OnCalcFields* non viene chiamato quando si modificano singoli campi all'interno di un record (la quarta condizione vista in precedenza).



OnCalcFields viene chiamato frequentemente, quindi il codice scritto per esso deve essere breve. Inoltre, se *AutoCalcFields* è **true**, *OnCalcFields* non deve svolgere operazioni che modificano il dataset (o un dataset collegato nel caso appartienga a una relazione master/detail), in quanto ciò questo può dar luogo a ricorsività. Per esempio, se *OnCalcFields* esegue un'operazione *Post* e *AutoCalcFields* è **true**, *OnCalcFields* viene chiamato nuovamente, dando origine a un'altra operazione *Post*, e così via.

Quando viene eseguito *OnCalcFields*, il dataset passa in modalità *dsCalcFields*. Questo stato impedisce modifiche o aggiunte ai record di un dataset, ad eccezione dei campi calcolati che il gestore è previsto possa modificare. Il motivo per prevenire altre modifiche è dovuto al fatto che *OnCalcFields* utilizza i valori contenuti in altri campi per derivare i valori dei campi calcolati. Eventuali modifiche a questi altri campi, potrebbero invalidare i valori assegnati ai campi calcolati. Dopo il completamento di *OnCalcFields*, il dataset torna allo stato *dsBrowse*.

Tipi di dataset

"Utilizzo dei discendenti di TDataSet" a pagina 22-2 classifica i discendenti di *TDataSet* in base al metodo che essi utilizzano per accedere ai propri dati. Un altro metodo utile per classificare i discendenti di *TDataSet* è quello di considerare il tipo di server dati che essi rappresentano. In base a quest'ottica, ci sono tre classi di base di dataset:

- **Datasets di tipo tabella**: I dataset di tipo tabella rappresentano una singola tabella di un server di database, comprese tutte le sue righe e colonne. $$$$I dataset di tipo tabella includono *TTable*, *TADOTable*, *TSQLTable* e *TIBTable*.

I dataset di tipo tabella permettono di sfruttare indici definiti sul server. Poiché c'è una corrispondenza biunivoca tra tabella di database e dataset, è possibile utilizzare gli indici server definiti per la tabella di database. Gli indici permettono all'applicazione di ordinare i record nella tabella, di velocizzare ricerche e consultazioni e possono costituire la base di una relazione master/detail. Inoltre, alcuni dataset di tipo tabella sfruttano il vantaggio della relazione uno-a-uno esistente tra il dataset e la tabella di database per permettere di eseguire operazioni a livello di tabella, come la creazione e la cancellazione di tabelle di database.

- **Dataset di tipo query** : I dataset di tipo query rappresentano un singolo comando SQL o una query. Le query possono rappresentare il set risultato derivante dall'esecuzione di un comando (di solito un'istruzione SELECT) o possono eseguire un comando che non restituisce alcun record (ad esempio, un'istruzione UPDATE). I dataset di tipo query includono *TQuery*, *TADOQuery*, *TSQLQuery* e *TIBQuery*.

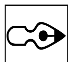
  Per utilizzare un dataset di tipo query, è necessario avere familiarità con SQL e con l'implementazione SQL del server, compresi eventuali limiti ed estensioni ampliamenti allo standard SQL-92. Se non si ha padronanza di SQL, potrebbe valer la pena acquistare un manuale tecnico che tratti SQL in dettaglio. Uno dei migliori è *Understanding the New SQL: A Complete Guide*, di Jim Melton e Alan R. Simpson, edito da Morgan Kaufmann.

- **Dataset di tipo procedura registrata**: I dataset di tipo procedura registrata rappresentano una procedura registrata sul server di database. I dataset di tipo procedura registrata includono *TStoredProc*, *TADOStoredProc*, *TSQLStoredProc* e *TIBStoredProc*.

  Una procedura registrata è un programma indipendente scritto nella linguaggio di procedure e trigger specifici del sistema di database utilizzato. Di solito gestiscono operazioni relative al database ripetute di frequente, e risultano particolarmente utili per operazioni che agiscono su grandi quantità di record o che utilizzano funzioni di aggregazione o matematiche. Di solito l'utilizzo di procedure registrate migliora le prestazioni di un'applicazione database:

  - Sfruttando l'enorme potenza e velocità di elaborazione tipiche dei server.
  - Riducendo il traffico di rete perché tutte le elaborazioni vengono spostate sul server.

  Le procedure registrate possono restituire o meno dei dati. Quelle che restituiscono dati possono restituirli come un cursor (in modo analogo al risultato di una query SELECT), come più cursor (restituendo in effetti più dataset) o restituire dati nei parametri di output. Queste differenze dipendono in parte dal server: alcuni server non permettono alle procedure registrate di restituire dati o permettono solo parametri di output. Altri server invece non supportano affatto le procedure registrate. Per determinare cosa è possibile fare, consultare la documentazione del server.



Solitamente è possibile utilizzare un dataset di tipo query per eseguire procedure registrate poiché la maggior parte dei server dispone di estensioni a SQL per operare con le procedure registrate. Ogni server, tuttavia, utilizza a questo scopo la propria

sintassi. Se si sceglie di utilizzare un dataset di tipo query invece di un dataset di tipo procedura registrata, consultare la documentazione del server per la sintassi necessaria.

Oltre ai dataset che rientrano chiaramente in queste tre categorie, *TDataSet* ha alcuni discendenti che possono essere considerati appartenenti a più di una categoria:

- *TADODataSet* e *TSQLDataSet* hanno una proprietà *CommandType* che permette di specificare se rappresentano una tabella, una query o una procedura registrata. I nomi delle proprietà e dei metodo sono molto simili a quelli dei dataset di tipo query, benché *TADODataSet* permetta di specificare un indice come un dataset di tipo tabella.
- *TClientDataSet* rappresenta i dati da un altro dataset. Come tale, può rappresentare una tabella, una query o una procedura registrata. *TClientDataSet* si comporta in modo molto simile a un dataset di tipo tabella, a causa del suo supporto per gli indici. Tuttavia, anche ha alcune delle funzioni dei dataset di tipo query e procedura registrata: la gestione dei parametri e la capacità di essere eseguito senza acquisire un set risultato.
- Alcuni altri dataset client (*TBDEClientDataSet* e *TSQLClientDataSet*) hanno una proprietà *CommandType* che permette di specificare se essi rappresentano una tabella, una query o una procedura registrata. I nomi delle proprietà e dei metodi sono come quelli di *TClientDataSet*, e gli somiglia anche per il supporto dei parametri e degli indici e la capacità la capacità di essere eseguito senza acquisire un set risultato.
- *TIBDataSet* può rappresentare sia le query che le procedure registrate. Infatti, può rappresentare più query e procedure registrate contemporaneamente, con proprietà separate per ognuno.

Utilizzo dei dataset di tipo tabella

Per utilizzare un dataset di tipo tabella,

1. Collocare il componente dataset appropriato in un modulo dati o su una scheda e impostarne la proprietà *Name* a un valore univoco, consono all’applicazione.
2. Identificare il server di database che contiene la tabella che si vuole usare. Ogni dataset di tipo tabella lo compie in modo diverso, ma di solito si specifica un componente database *connection*:
   - Per *TTable*, specificare un componente *TDatabase* o un alias BDE utilizzando la proprietà *DatabaseName*.
   - Per *TADOTable*, specificare un componente *TADOConnection* utilizzando la proprietà *Connection*.
   - Per *TSQLTable*, specificare un componente *TSQLConnection* utilizzando la proprietà *SQLConnection*.
   - Per *TIBTable*, specificare un componente *TIBConnection* utilizzando la proprietà *Database*.

Per informazioni sull'utilizzo di componenti database *connection*, vedere il Capitolo 21 Capitolo 21, "Connessione ai database".

3 Impostare la proprietà al nome della tabella nel database. È possibile selezionare le tabelle da una lista a discesa se si è già identificato un componente database *connection*.

4 Mettere un componente *datasource* nel modulo dati o sulla scheda e impostarne la proprietà *DataSet* al nome del dataset. Il componente *datasource* viene usato per passare un *set* risultato dal dataset ai componenti data-aware affinché siano visualizzati.

Vantaggi dell'utilizzo dei dataset di tipo tabella

Il vantaggio principale dell'utilizzo di dataset di tipo tabella è la disponibilità di indici. Gli indici consentono all'applicazione di

- Ordinare i record nel dataset.
- Individuare rapidamente i records.
- Limitare i record visibili.
- Stabilire relazioni master/detail.

Inoltre, la relazione uno-a-uno tra dataset di tipo tabella e tabelle di database permette di utilizzare molti di loro per i seguenti scopi

- Controllo dell'accesso alle tabelle in lettura/scrittura
- Creazione e cancellazione di tabelle
- Svuotamento di tabelle
- Sincronizzazione di tabelle

Ordinamento dei record con indici

Un indice determina l'ordine di visualizzazione dei record in una tabella. Di solito, i record vengono visualizzati in ordine ascendente in base a un indice primario o predefinito. Questo comportamento predefinito non richiede intervento da parte dell'applicazione. Se si desidera una sequenza di ordinamento diversa, tuttavia, è necessario specificare

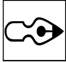
- Un indice alternativo, oppure.
- Una lista di colonne su cui ordinare (non disponibile su server che non sono basati su SQL).

Gli indici permettono di presentare i dati da una tabella secondo diversi ordinamenti. Su tabelle basate su SQL, questa sequenza di ordinamento è implementata utilizzando l'indice per generare una clausola ORDER BY in una query che preleva i record della tabella. Su altre tabelle (come tabelle Paradox e dBASE), l'indice è utilizzato dal meccanismo di accesso di dati per presentare i record nell'ordine richiesto.

Ottenimento di informazioni sugli indici

L'applicazione può ottenere informazioni su indici definiti dal server da tutti i dataset di tipo tabella. Per ottenere una lista degli indici disponibili per il dataset, chiamare il metodo GetIndexName. *GetIndexNames* riempie una lista di stringhe con i nomi degli indici validi. Ad esempio, il seguente codice riempie una casella di riepilogo con i nomi di tutti gli indici definiti per il dataset *CustomersTable*:

```
CustomersTable->GetIndexNames(ListBox1->Items);
```



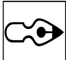
Nel caso di tabelle Paradox, l'indice principale è senza nome e non viene quindi restituito da *GetIndexNames*. Con una tabella Paradox, comunque, è sempre possibile reimpostare l'indice in uso all'indice principale, dopo avere utilizzato un indice alternativo, impostando la proprietà *IndexName* a una stringa vuota.

Per ottenere informazioni sui campi dell'indice corrente, utilizzare

- La proprietà *IndexFieldCount*, per determinare il numero di colonne nell'indice.
- La proprietà *IndexFields*, per esaminare una lista dei componenti campo per le colonne che comprendono l'indice.

Il codice seguente mostra come si protrebbero utilizzare *IndexFieldCount* e *IndexFields* per passare in sequenza una lista di nomi di colonna in un'applicazione:

```
AnsiString ListOfIndexFields[20];
for (int i = 0; i < CustomersTable->IndexFieldCount; i++)
  ListOfIndexFields[i] = CustomersTable->IndexFields[i]->FieldName;
```



IndexFieldCount non è valido per una tabella dBASE aperta in base a un indice di espressione.

Specifica di un indice con IndexName

Per fare in modo che un indice diventi attivo, utilizzare la proprietà *IndexName*. Una volta attivo, un indice determina l'ordine dei record nel dataset. (Può essere utilizzato anche come base per un collegamento master-detail, per una ricerca basata su indice o per un filtraggio basato su indice).

Per attivare un indice, impostare la proprietà *IndexName* al nome di un indice. In alcuni sistemi di database, gli indici principali non hanno nomi. Per attivare uno di questi indici, impostare *IndexName* a una stringa vuota.

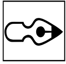
In progettazione, è possibile selezionare un indice da una lista di indici disponibili facendo clic nell'Object Inspector sul pulsante ellissi della proprietà. In fase di esecuzione, impostare *IndexName* utilizzando una costante o una variabile *AnsiString*. È possibie ottenere una lista degli indici disponibili chiamando il metodo *GetIndexName*.

Il seguente codice imposta l'indice per *CustomersTable* a *CustDescending*:

```
CustomersTable->IndexName = "CustDescending";
```

Creazione di un indice con IndexFieldNames

Se non c'è alcun indice definito che implementa la sequenza di ordinamento desiderata, è possibile creare uno pseudo-indice utilizzando la proprietà *IndexFieldNames*.

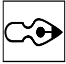


IndexName and *IndexFieldNames* sono reciprocamente esclusivi. L'impostazione della proprietà di uno cancella i valori impostati per l'altro.

Il valore di *IndexFieldNames* è una stringa ansi. Per specificare una sequenza di ordinamento, elencare ogni nome di colonna da utilizzare nell'ordine che dovrebbe essere utilizzato e delimitare i nomi con punti e virgola. L'ordinamento è solo ascendente. La dipendenza dell'ordinamento dalle maiuscole è legata alle capacità del server. Per ulteriori informazioni, vedere la documentazione del server.

Il seguente codice imposta la sequenza di ordinamento per *PhoneTable* in base a *LastName*, quindi a *FirstName*:

```
PhoneTable->IndexFieldNames = "LastName;FirstName";
```



Se si utilizza *IndexFieldNames* su tabelle Paradox e dBASE, il dataset tenta di trovare un indice che utilizza le colonne specificate. Se non è in grado di trovare un tale indice, solleva un'eccezione.

Utilizzo di indici per la ricerca di record

Si può effettuare una ricerca in un qualsiasi dataset usando i metodi *Locate* e *Lookup* di *TDataSet*. Comunque, se si utilizzano esplicitamente gli indici, alcuni dataset di tipo tabella possono migliorare di molto le prestazioni di una ricerca fornite dai metodi *Locate* e *Lookup*.

I dataset ADO supportano tutti il metodo *Seek*, che sposta a un record in base a un insieme di valori di campo per i campi nell'indice corrente. *Seek* permette di specificare dove spostare il cursore rispetto al primo o all'ultimo record corrispondente.

TTable e tutti i tipi di dataset client supportano ricerche simili basate su indici, ma utilizzano una combinazione di metodi collegati. La seguente tabella riassume i sei metodi collegati forniti da *TTable* e dai dataset client per supportare le ricerche basate su indice:

Tabella 22.9 Metodi di ricerca basati sugli indici

Metodo	Scopo
EditKey	Preserva il contenuto corrente del buffer della chiave di ricerca e pone il dataset nello stato *dsSetKey* in modo che l'applicazione possa modificare i criteri di ricerca esistenti prima dell'esecuzione di una ricerca.
FindKey	Combina in un singolo metodo i metodi *SetKey* e *GotoKey*.
FindNearest	Combina in un singolo metodo i metodi *SetKey* e *GotoNearest*.
GotoKey	Ricerca il primo record in un dataset che corrisponde esattamente ai criteri di ricerca e, nel caso lo trovi, sposta il cursore su quel record.
GotoNearest	Ricerca, in base a un valore di chiave parziale, nei campi basati su stringhe la corrispondenza più simile a un record e sposta il cursore a quel record.
SetKey	Cancella il buffer della chiave di ricerca e pone la tabella nello stato *dsSetKey* in modo che l'applicazione possa specificare nuovi criteri di ricerca prima di eseguire una ricerca.

GotoKey e *FindKey* sono funzioni boolean che, se vanno a buon fine, spostano il cursore a un record corrispondente e restituiscono **true**. Se la ricerca non va a buon fine, il cursore non viene spostato e queste funzioni restituiscono **false**.

GotoNearest e *FindNearest* ricollocano sempre il cursore o sulla prima corrispondenza esatta trovata o, nel caso non venga trovata alcuna corrispondenza, sul primo record che è maggiore dei criteri di ricerca specificati.

Esecuzione di una ricerca con metodi Goto

Per eseguire una ricerca utilizzando i metodi *Goto*, seguire questi passi generali:

1 Specificare l'indice da utilizzare per la ricerca. Questo è lo stesso indice che ordina i record nel dataset (vedere "Ordinamento dei record con indici" a pagina 22-27). Per specificare l'indice, utilizzare la proprietà *IndexName* o *IndexFieldNames*.
2 Aprire il dataset.
3 Porre il dataset nello stato *dsSetKey* chiamando *SetKey*.
4 Specificare il valore o i valori da ricercare nella proprietà *Fields*. *Fields* è un oggetto *TFields*, che gestisce una lista indicizzata di componenti campo, a cui è possibile accedere specificando il numero ordinale che corrisponde alla colonna. Il primo numero di colonna in un dataset è 0.
5 Cercare e spostarsi al primo record corrispondente trovato con *GotoKey* o con *GotoNearest*.

Ad esempio, il codice seguente, collegato all'evento *OnClick* di un pulsante, utilizza il metodo *GotoKey* per spostarsi al primo record in cui il primo campo nell'indice ha un valore che corrisponde esattamente al testo immesso in una casella di testo:

```
void __fastcall TSearchDemo::SearchExactClick(TObject *Sender)
{
  ClientDataSet1->SetKey();
  ClientDataSet1->Fields->Fields[0]->AsString = Edit1->Text;
  if (!ClientDataSet1->GotoKey())
    ShowMessage("Record not found");
}
```

GotoNearest è simile. Cerca la corrispondenza più vicina a un valore parziale di un campo. Può essere utilizzato solo per campi stringa. Per esempio,

```
Table1->SetKey();
Table1->Fields->Fields[0]->AsString = "Sm";
Table1->GotoNearest();
```

Se esiste un record che contiene "Sm" come primi due caratteri del valore del primo campo indicizzato, il cursore viene posizionato su quel record. Altrimenti, la posizione del cursore non cambia e *GotoNearest* restituisce **false**.

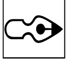
Esecuzione di una ricerca con metodi Find

I metodi *Find* fanno la stessa cosa dei metodi *Goto*, salvo che non è necessario porre esplicitamente il dataset nello stato *dsSetKey* per specificare i valori del campo chiave su cui effettuare la ricerca. Per eseguire una ricerca utilizzando i metodi *Find*, seguire questi passi generali:

1 Specificare l'indice da utilizzare per la ricerca. Questo è lo stesso indice che ordina i record nel dataset (vedere "Ordinamento dei record con indici" a pagina 22-27). Per specificare l'indice, utilizzare la proprietà *IndexName* o *IndexFieldNames*.

2 Aprire il dataset.

3 Ricercare e spostarsi al primo record, o a quello più vicino, con *FindKey* o con *FindNearest*. Entrambi i metodi richiedono un singolo parametro, una lista di valori di campo, delimitata con virgole, in cui ogni valore corrisponde a una colonna indicizzata della tabella sottostante.



FindNearest può essere utilizzato solo per campi stringa.

Specifica del record corrente dopo una ricerca andata a buon fine

Per impostazione predefinita, una ricerca che va a buon fine posiziona il cursore sul primo record che corrisponde ai criteri di ricerca. Se si preferisce, è possibile impostare la proprietà *KeyExclusive* a **true** in modo da posizionare il cursore sul record immediatamente successivo al primo record corrispondente.

Per impostazione predefinita, *KeyExclusive* è **false**, vale a dire che una ricerca andata a buon fine posiziona il cursore sul primo record corrispondente.

Ricerca in base a chiavi parziali

Se un dataset client ha più di una colonna chiave e si vuole effettuare una ricerca per valori in un sottoinsieme di quella chiave, impostare *KeyFieldCount* al numero di colonne su cui si sta facendo la ricerca. Ad esempio, se l'indice corrente del dataset client ha tre colonne e si vogliono cercare dei valori utilizzando solo la prima colonna, impostare *KeyFieldCount* a 1.

Nel caso di dataset ditpo tabella con chiavi su più colonne, è possibile cercare i valori solo in colonne contigue, iniziando dalla prima. Ad esempio, per una chiave composta da tre colonne è possibile cercare i valori nella prima colonna, nella prima e nella seconda o nella prima, seconda e terza ma non solo nella prima e nella terza.

Ripetizione o ampliamento di una ricerca

Ogni volta che si chiamano *SetKey* o *FindKey*, il metodo azzera qualsiasi valore precedente nella proprietà *Fields*. Se si desidera ripetere una ricerca utilizzando i campi impostati in precedenza o si desidera fare un'aggiunta ai campi utilizzati in una ricerca, chiamare *EditKey* invece di *SetKey* e di *FindKey*.

Ad esempio, si supponga di aver già eseguita una ricerca nella tabella Employee in base al campo *City* dell'indice "CityIndex". Si supponga inoltre che "CityIndex" includa anche i campi *City* e *Company*. Per trovare un record con un determinato nome di azienda in una determinata città, utilizzare il seguente codice:

```
Employee->KeyFieldCount = 2;
Employee->EditKey();
Employee->FieldValues["Company"] = Variant(Edit2->Text);
Employee->GotoNearest();
```

Limitazione dei record con gli intervalli

È possibile vedere e modificare temporaneamente un sottoinsieme di dati di qualsiasi dataset usando i filtri (consultare "Visualizzazione e modifica di un sottoinsieme di dati usando i filtri" a pagina 22-13). Alcuni dataset di tipo tabella supportano anche un altro modo per accedere a un sottoinsieme dei record disponibili, detto intervalli.

Gli intervalli sono utilizzabili sono con *TTable* e con i dataset client. Malgrado le loro somiglianze, gli intervalli e i filtri hanno utilizzi differenti. I paragrafi seguenti descrivono le differenze fra intervalli e filtri e spiegano come utilizzare gli intervalli.

Differenze fra intervalli e filtri

Sia gli intervalli che i filtri limitano i record visibili a un sottoinsieme di tutti i record disponibili, ma il loro modo di operare è diverso. Un intervallo è un insieme di record indicizzati in contiguo che sono compresi entro valori limite specificati. Ad esempio, in un database di dipendenti indicizzato in base al cognome, si potrebbe applicare un intervallo per visualizzare tutti i dipendenti i cui cognomi sono maggiori di "Jones" e minori di "Smith". Poiché gli intervalli dipendono dagli indici, è necessario impostare come indice corrente quello che può essere utilizzato per definire l'intervallo. Analogamente alla specifica di un indice per ordinare i record, utilizzando la proprietà *IndexName* oppure *IndexFieldNames* è possibile assegnare l'indice in base a cui definire un intervallo.

Un filtro, invece, è un qualunque insieme di record che condividono dei punti dati specificati, indipendentemente dalla loro indicizzazione. Ad esempio, si potrebbe filtrare un database di dipendenti per visualizzare tutti i dipendenti che vivono in California e che hanno lavorato per la società per cinque o più anni. Benché i filtri possono fare uso di indici nel caso siano disponibili, i filtri non dipendono dagli indici. I filtri vengono applicati record per record mentre un'applicazione esamina in sequenza un dataset.

Di solito, i filtri sono più flessibili degli intervalli. Gli intervalli, tuttavia, possono essere più efficienti quando si opera con dataset molto grandi e i record che interessano a un'applicazione sono già bloccati in gruppi indicizzati in modo contiguo. Per dataset molto grandi, potrebbe risultare ancora più efficiente l'utilizzo della clausola WHERE di un dataset di tipo query per selezionare i dati. Per i dettagli sulla specifica di una query, vedere "Utilizzo di dataset di tipo query" a pagina 22-43.

Specifica degli intervalli

Esistono due modi reciprocamente esclusivi per specificare un intervallo:

- Specificare l'inizio e la fine utilizzando separatamente *SetRangeStart* e *SetRangeEnd*.
- Specificare entrambi i punti terminali in una sola volta utilizzando *SetRange*.

Impostazione dell'inizio di un intervallo

Chiamare la procedura *SetRangeStart* per porre il dataset nello stato *dsSetKey* e iniziare a creare una lista di valori iniziali per l'intervallo. Una volta che si chiama *SetRangeStart*, le successive assegnazioni alla proprietà *Fields* sono trattate come

valori iniziali dell'indice da utilizzare nell'applicazione dell'intervallo. I campi specificati devono essere inclusi nell'indice corrente.

Ad esempio, si supponga che l'applicazione utilizzi un componente *TSQLClientDataSet* di nome *Customers*, collegato con la tabella CUSTOMER e che siano stati creati dei componenti campo persistenti per ogni campo nel dataset *Customers*. CUSTOMER è indicizzato sulla prima colonna (*CustNo*). Una finestra dell'applicazione consente di modificare due componenti di nome *StartVal* e *EndVal*, utilizzati per specificare il valore iniziale e finale di un intervallo. Per creare e applicare un intervallo si può utilizzare il seguente codice:

```
Customers->SetRangeStart();
Customers->FieldValues["CustNo"] = StrToInt(StartVal->Text);
Customers->SetRangeEnd();
if (!EndVal->Text.IsEmpty())
  Customers->FieldValues["CustNo"] = StrToInt(EndVal->Text);
Customers->ApplyRange();
```

Prima di assegnare un qualsiasi valore a *Fields*, questo codice controlla che il testo immesso in *EndVal* non sia nullo. Se il testo immesso per *StartVal* è nullo, allora vengono inclusi tutti i record dall'inizio del dataset, perché tutti i valori sono maggiori di zero. Tuttavia, se il testo immesso per *EndVal* è nullo, allora non viene incluso nessun record, perché nessun record è minore di nullo.

Nel caso di un indice su più colonne, è possibile specificare un valore iniziale per tutti o per alcuni campi nell'indice. Se non si fornisce il valore di un campo usato nell'indice, quando si applica l'intervallo viene assunto un valore nullo. Se si prova a impostare un valore per un campo che non è incluso nell'indice, il dataset solleva un'eccezione.



Per partire dall'inizio del dataset, omettere la chiamata a *SetRangeStart*.

Per terminare di specificare l'inizio di un intervallo, chiamare *SetRangeEnd* oppure applicare o annullare l'intervallo. Per informazioni sull'applicazione e l'annullamento di intervalli, consultare "Applicazione o annullamento di un intervallo" a pagina 22-36.

Impostazione della fine di un intervallo

Chiamare la procedura *SetRangeEnd* per mettere il dataset nello stato *dsSetKey* e cominciare a creare una lista di valori finali per l'intervallo. Una volta chiamata *SetRangeEnd*, le successive assegnazioni alla proprietà *Fields* vengono trattate come valori finali di indice da usare quando si applicherà l'intervallo. I campi specificati devono essere inclusi nell'indice corrente.



Occorre specificare sempre i valori finali di un intervallo, anche se si vuole che l'intervallo finisca all'ultimo record del dataset. Se non si forniscono valori finali, C++Builder presuppone che il valore finale dell'intervallo sia un valore nullo. Un intervallo con valori finali nulli è sempre vuoto.

Il modo più facile per assegnare i valori finali è quello di chiamare il metodo *FieldByName* method. Per esempio,

```
Contacts->SetRangeStart();
Contacts->FieldByName("LastName")->Value = Edit1->Text;
```

```
Contacts->SetRangeEnd();
Contacts->FieldByName("LastName")->Value = Edit2->Text;
Contacts->ApplyRange();
```

Come per la specifica dei valori iniziali dell'intervallo, se si prova a impostare un valore per un campo che non è incluso nell'indice, il dataset solleva un'eccezione.

Per terminare di specificare la fine di un intervallo, applicare o annullare l'intervallo. Per informazioni sull'applicazione e l'annullamento di intervalli, consultare "Applicazione o annullamento di un intervallo" a pagina 22-36.

Impostazione del valore iniziale e finale di un intervallo

Invece di utilizzare chiamate separate a *SetRangeStart* e a *SetRangeEnd* per specificare i limiti dell'intervallo, è possibile chiamare la procedura *SetRange* per porre il dataset nello stato *dsSetKey* e impostare i valori iniziali e finali di un intervallo con una singola chiamata.

SetRange richiede due parametri constant array: una serie di valori iniziali e una serie di valori finali. Le istruzioni seguenti, ad esempio, stabiliscono un intervallo basato su un indice a due colonne:

```
TVarRec StartVals[2];
TVarRec EndVals[2];
StartVals[0] = Edit1->Text;
StartVals[1] = Edit2->Text;
EndVals[0] = Edit3->Text;
EndVals[1] = Edit4->Text;

Table1->SetRange(StartVals, 1, EndVals, 1);
```

Nel caso di un indice su più colonne è possibile specificare i valori di inizio e di fine per tutti i campi dell'indice o solo per alcuni di essi. Se non si fornisce il valore di un campo usato nell'indice, quando si applica l'intervallo viene assunto un valore nullo. Per omettere un valore per il primo campo in un indice e specificare dei valori per i campi consecutivi, passare un valore nullo per il campo omesso.

Occorre specificare sempre i valori finali di un intervallo, anche se si vuole che l'intervallo finisca all'ultimo record del dataset. Se non si forniscono valori finali, il dataset presuppone che il valore finale dell'intervallo sia un valore nullo. Un intervallo con valori finali nulli è sempre vuoto perché l'intervallo iniziale è maggiore o uguale all'intervallo finale.

Specifica di un intervallo in base a chiavi parziali

Se una chiave è composta da uno o più campi di tipo stringa, i metodi *SetRange* supportano chiavi parziali. Se, ad esempio, un indice è basato sulle colonne *LastName* e *FirstName*, le seguenti specifiche di intervallo risultano valide:

```
Contacts->SetRangeStart();
Contacts->FieldValues["LastName"] = "Smith";
Contacts->SetRangeEnd();
Contacts->FieldValues["LastName"] = "Zzzzzz";
Contacts->ApplyRange();
```

Questo codice include tutti i record in un intervallo in cui *LastName* è maggiore o uguale a ″Smith″. La specifica del valore potrebbe anche essere:

```
Contacts->FieldValues["LastName"] = "Sm";
```

Questa istruzione include i record in cui *LastName* è maggiore o uguale a ″Sm″.

Inclusione o esclusione dei record che corrispondono ai limiti dell'intervallo

Per impostazione predefinita, un intervallo include tutti i record maggiori o uguali al valore di inizio intervallo e minori o uguali al valore di fine intervallo specificati. Questo comportamento è controllato dalla proprietà *KeyExclusive*. Per impostazione predefinita *KeyExclusive* è **false**.

Se si preferisce, è possibile impostare la proprietà *KeyExclusive* di un dataset a **true** in modo da escludere i record uguali al valore di fine intervallo. Ad esempio,

```
Contacts->SetRangeStart();
Contacts->KeyExclusive = true;
Contacts->FieldValues["LastName"] = "Smith";
Contacts->SetRangeEnd();
Contacts->FieldValues["LastName"] = "Tyler";
Contacts->ApplyRange();
```

Questo codice include tutti i record in un intervallo in cui *LastName* è maggiore o uguale a ″Smith″ e minore di ″Tyler″.

Modifica di un intervallo

Due funzioni permettono di modificare le condizioni esistenti dei limiti estremi di un intervallo: *EditRangeStart*, per modificare i valori iniziali di un intervallo; ed *EditRangeEnd*, per modificare i valori finali dell'intervallo.

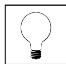
Il processo per modificare e applicare un intervallo implica questi passi generali:

1. Porre il dataset nello stato *dsSetKey* e modificare il valore iniziale dell'indice dell'intervallo.
2. Modifica del valore finale dell'indice dell'intervallo.
3. Applicare l'intervallo al dataset.

È possibile modificare o il singolo valore iniziale o finale dell'intervallo, oppure si possono modificare entrambe le condizioni dei limiti estremo. Se si modificano le condizioni relative ai limiti estremi di un intervallo che è attualmente applicato a un dataset, le modifiche apportate non divengono effettive finché non si chiama nuovamente *ApplyRange*.

Modifica dell'inizio di un intervallo

Chiamare la procedura *EditRangeStart* per porre il dataset nello stato *dsSetKey* e iniziare a modificare la lista corrente dei valori iniziali dell'intervallo. Una volta che si chiama *EditRangeStart*, le successive assegnazioni alla proprietà *Fields* sovrascrivono i valori correnti di indice da utilizzare al momento dell'applicazione dell'intervallo.



Se inizialmente si era creato un inizio intervallo in base a una chiave parziale, è possibile utilizzare *EditRangeStart* per ampliare il valore iniziale dell'intervallo. Per ulteriori informazioni sugli intervalli basati su chiavi parziali, consultare "Specifica di un intervallo in base a chiavi parziali" a pagina 22-34.

Modifica della fine di un intervallo

Chiamare la procedura *EditRangeEnd* per porre il dataset nello stato *dsSetKey* e iniziare a creare una lista di valori finali per l'intervallo. Una volta che si chiama *EditRangeEnd*, le successive assegnazioni alla proprietà *Fields* sono trattate come valori finali dell'indice da utilizzare al momento dell'applicazione dell'intervallo.

Applicazione o annullamento di un intervallo

Quando si chiamano *SetRangeStart* o *EditRangeStart* per specificare l'inizio di un intervallo, oppure *SetRangeEnd* o *EditRangeEnd* per specificare la fine di un intervallo, il dataset viene posto nello stato *dsSetKey*. Esso resta in quello stato finché non si applica o non si annulla l'intervallo.

Applicazione di un intervallo

Quando si specifica un intervallo, le condizioni di limite che vengono definite non diventano effettive finché non si applica l'intervallo. Affinché un intervallo entri in vigore, si deve chiamare il metodo *ApplyRange*. *ApplyRange* limita immediatamente la vista e l'accesso da parte di un utente solo ai dati inclusi nel sottoinsieme specificato del dataset.

Annullamento di un intervallo

Il metodo *CancelRange* pone fine all'applicazione di un intervallo e ripristina l'accesso all'intero dataset. Anche se l'annullamento un intervallo ripristina l'accesso a tutti i record nel dataset, le condizioni di limite di quell'intervallo sono ancora disponibili, rendendo possibile riapplicare l'intervallo in un momento successivo. I limiti di intervallo vengono conservati finché non si forniscono nuovi limiti per intervallo o non si modificano i limiti esistenti. Ad esempio, il seguente codice è valido:

```
⋮
MyTable->CancelRange();
⋮
// later on, use the same range again. No need to call SetRangeStart, etc.
MyTable->ApplyRange();
⋮
```

Creazione di relazioni master/detail

I dataset di tipo tabella possono essere collegati mediante relazioni master/detail. Quando si imposta una relazione master/detail, si collegano due dataset in modo che tutti i record di uno (il detail) corrispondono sempre al singolo record attivo dell'altro (il master).

I dataset di tipo tabella supportano le relazioni master/detail in due modi molto diversi:

- Tutti i dataset di tipo tabella possono agire come dettaglio di un altro dataset mediante il collegamento dei cursor. Questo processo è descritto in "Trasformazione della tabella nella tabella dettaglio di un altro dataset".
- *TTable*, *TSQLTable* e tutti i dataset client possono agire come master in una relazione di master/detail che utilizza tabelle dettaglio annidate. Questo processo viene descritto nella sezione "Utilizzo di tabelle dettaglio annidate" a pagina 22-38.

Ognuno di questi approcci offre i propri vantaggi. Il collegamento di cursor permette di creare relazioni master/detail in cui la tabella principale è un qualsiasi tipo di dataset. Con tabelle di dettaglio annidate, i tipi di dataset che possono agire come tabella di dettaglio sono limitati, ma offrono molte più possibilità nel modo di visualizzare i dati. Se il master è un dataset client, le tabelle dettaglio annidate forniscono un meccanismo più affidabile per applicare gli aggiornamenti in cache.

Trasformazione della tabella nella tabella dettaglio di un altro dataset

Le proprietà *MasterSource* e *MasterFields* di un dataset di tipo tabella possono essere utilizzate per impostare relazioni uno a molti tra due dataset

La proprietà *MasterSource* è utilizzata per specificare un datasource da cui la tabella ottiene i dati dalla tabella principale. Questo datasource può essere collegato con qualsiasi tipo di dataset. Per esempio, se in questa proprietà si specifica il datasource di una query, è possibile collegare un dataset client come dettaglio delle query, in modo che il dataset client tracci gli eventi che si verificano nella query.

Il dataset è collegato con la tabella principale in base al suo indice corrente. Prima di specificare i campi del dataset principale che vengono tracciati dal dataset di dettaglio, specificare per prima cosa l'indice nel dataset di dettaglio che inizia con i campi corrispondenti. È possibile utilizzare le proprietà *IndexName* o *IndexFieldNames*.

Una volta specificato l'indice da utilizzare, utilizzare la proprietà *MasterFields* per indicare la o le colonne nel dataset principale che corrispondono ai campi dell'indice nella tabella di dettaglio). Per collegare dataset in base a più nomi di colonne, separare i nomi dei campi con punti e virgola:

```
Parts->MasterFields = "OrderNo;ItemNo";
```

Per aiutare a creare collegamenti significativi tra i due dataset, è possibile utilizzare Field Link designer. Per utilizzare Field Link designer, fare doppio clic nell'Object Inspector sulla proprietà *MasterFields* dopo avere assegnato un *MasterSource* e un indice.

Le istruzioni successive consentono di creare una semplice scheda in cui l'utente può esaminare ad uno ad uno i record dei clienti e visualizzare tutti gli ordini relativi al cliente corrente. La tabella master è la tabella *CustomersTable* e la tabella dettaglio è *OrdersTable*. L'esempio utilizza il componente basato su BDE *TTable*, ma è possibile utilizzare gli stessi metodi per collegare qualsiasi dataset di tipo tabella.

1. Collocare in un modulo dati due componenti *TTable* e due componenti *TDataSource*.
2. Impostare le proprietà del primo componente *TTable* come segue:

- *DatabaseName*: BCDEMOS
- *TableName*: CUSTOMER
- *Name*: CustomersTable

3 Impostare le proprietà del secondo componente *TTable* come segue:

- *DatabaseName*: BCDEMOS
- *TableName*: ORDERS
- *Name*: OrdersTable

4 Impostare le proprietà del primo componente *TDataSource* come segue:

- *Name*: CustSource
- *DataSet*: CustomersTable

5 Impostare le proprietà del secondo componente *TDataSource* come segue:

- *Name*: OrdersSource
- *DataSet*: OrdersTable

6 Collocare su una scheda due componenti *TDBGrid*.

7 Scegliere l'opzione File|Include Unit Hdr per specificare che la scheda deve utilizzare il modulo dati.

8 Impostare la proprietà *DataSource* del primo componente griglia a "CustSource", e impostare la proprietà *DataSource* della seconda griglia a "OrdersSource".

9 Impostare la proprietà *MasterSource* di *OrdersTable* a "CustSource". In questo modo la tabella CUSTOMER (la tabella principale) e la tabella ORDERS (la tabella di dettaglio) risultano collegate.

10 Fare doppio clic nell'Object Inspector sulla casella del valore della proprietà *MasterFields* in modo da richiamare il Field Link Designer e impostare le seguenti proprietà:

- Nel campo Available Indexes, scegliere *CustNo* per collegare le due tabelle in base al campo *CustNo*.
- Selezionare *CustNo* in entrambe le liste dei campi Detail Fields e Master Fields.
- Fare clic sul pulsante Add per aggiungere questa condizione di join. Nell'elenco Joined Fields, apparirà la scritta "CustNo -> CustNo".
- Scegliere OK per confermare le selezioni e uscire da Field Link Designer.

11 Impostare le proprietà *Active* di *CustomersTable* e *OrdersTable* a **true** per visualizzare i dati nelle griglie della scheda.

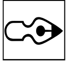
12 Compilare ed eseguire l'applicazione.

Se si esegue ora l'applicazione, si potrà notare che le tabelle sono collegate e che, quando ci si sposta su un nuovo record della tabella CUSTOMER, nella tabella ORDERS si vedono solo quei record che appartengono al cliente corrente.

Utilizzo di tabelle dettaglio annidate

Una tabella annidata è un dataset dettaglio che è il valore di un singolo campo di dataset in un altro dataset (master). Per i dataset che rappresentano dati del server,

un dataset dettaglio annidato può essere utilizzato solo per un campo di dataset sul server. I componenti *TClientDataSet* non rappresentano dati del server, ma, se per loro si crea un dataset che contiene dettagli annidati o se ricevono dati da un provider collegato con la tabella master di una relazione di master/detail, possono anche contenere campi di dataset.



Per *TClientDataSet*, l'uso di set di dettaglio annidati è necessario se si vogliono applicare gli aggiornamenti dalle tabelle master e detail al server di database.

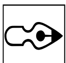
Per utilizzare i *set* di dettaglio annidati, la proprietà *ObjectView* del dataset master deve essere impostata a **true**. Quando il dataset di tipo tabella contiene i dataset di dettaglio annidati, *TDBGrid* fornisce il supporto per la visualizzazione dei dettagli annidati in una finestra a comparsa. Per ulteriori informazioni sul funzionamento, consultare "Visualizzazione dei campi dataset" a pagina 23-27.

In alternativa, è possibile visualizzare e modificare i dataset di dettaglio in controlli associati ai dati utilizzando un componente dataset separato per il *set* di dettaglio. In progettazione, si devono creare campi persistenti per i i campi del dataset (master), utilizzando Fields Editor: fare clic destro sul dataset master e scegliere Fields Editor. Aggiungere al dataset un nuovo campo esistente facendo clic destro e scegliendo Add Fields. Definire il nuovo campo con il tipo DataSet Field. Nel Fields Editor definire la struttura della tabella di dettaglio. È necessario anche aggiungere campi persistenti per tutti gli altri campi utilizzati nel dataset master.

Il componente dataset per la tabella dettaglio è un discendente di dataset di uno dei tipi consentito dalla tabella master. Solo i componenti *TTable* permettono di usare componenti *TNestedDataSet* come dataset annidati. I componenti *TSQLTable* permettono l'uso di altri componenti *TSQLTable*. I componenti *TClientDataset* permettono l'uso di altri dataset client. Scegliere nella Component palette un dataset di tipo appropriato e aggiungerlo alla scheda o al modulo dati. Impostare la proprietà *DataSetField* di questo dataset di dettaglio al campo persistente *DataSet* nel dataset master. Infine, collocare sulla scheda o sul modulo dati un componente *datasource* e impostarne la proprietà *DataSet* al dataset dettaglio. I controlli data-aware possono utilizzare questo *datasource* per accedere ai dati nel *set* di dettaglio.

Controllo dell'accesso alle tabelle in lettura/scrittura

Per impostazione predefinita, quando si apre un dataset di tipo tabella, esso richiede i diritti di lettura e scrittura per la tabella di database sottostante. A seconda delle caratteristiche della tabella di database sottostante, il privilegio di scrittura richiesto potrebbe non essere concesso (ad esempio, quando si richiede l'accesso in scrittura a una tabella *SQL* su un server remoto e il server limita l'accesso alla tabella alla sola lettura).

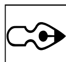


Questo non è vero per *TClientDataSet*, che determina se gli utenti possono modificare i dati provenienti da informazioni che il provider di dataset fornisce mediante pacchetti dati. Anche per *TSQLTable*, che è un dataset unidirezionale e perciò sempre a sola lettura, non è vero.

Quando si apre la tabella, è possibile controllare la proprietà *CanModify* per accertare se il database sottostante (o il provider di dataset) permette agli utenti di modificare i

dati nella tabella. Se *CanModify* è **false**, l'applicazione non può scrivere nel database. Se *CanModify* è **true**, l'applicazione può scrivere nel database a condizione che la proprietà *ReadOnly* della tabella sia **false**.

ReadOnly determina se un utente può sia visualizzare che modificare i dati. Quando *ReadOnly* è **false** (impostazione predefinita), un utente può sia vedere sia modificare i dati. Per limitare un utente alla sola visualizzazione dei dati, *ReadOnly* va impostata a **true** prima di aprire una tabella.



ReadOnly è implementata su tutti i dataset di tipo tabella tranne *TSQLTable*, che è sempre a sola lettura.

Creazione e cancellazione di tabelle

Alcuni dataset di tipo tabella permettono di creare e di cancellare le tabelle sottostanti in fase di progettazione o di esecuzione. Di solito, le tabelle di database vengono create e cancellate da un amministratore di database. Tuttavia, può essere utile durante sviluppo e il collaudo di applicazioni creare e distruggere le tabelle di database che l'applicazione può utilizzare.

Creazione di tabelle

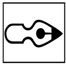
TTable e *TIBTable* permettono entrambi di creare la tabella di database sottostante senza utilizzare *SQL*. Allo stesso modo, *TClientDataSet* permette di creare un dataset quando non si sta operando con un provider di dataset. Utilizzando *TTable* e *TClientDataSet*, è possibile creare la tabella in progettazione o in esecuzione. Solo *TIBTable* permette di creare tabelle in esecuzione.

Prima di potere creare la tabella, è necessario aver impostato alcune proprietà per specificare la struttura della tabella che si vuole creare. In particolare, è necessario specificare:

- Il database che conterrà la nuova tabella. Per *TTable*, si specifica il database utilizzando la proprietà *DatabaseName*. Per *TIBTable*, è necessario utilizzare un componente *TIBDatabase*, assegnato alla proprietà *Database*. (I dataset client non utilizzano un database).
- Il tipo del database (solo per *TTable*). Impostare la proprietà *TableType* al tipo di tabella richiesto. Per tabelle Paradox, dBASE o ASCII, impostare rispettivamente *TableType* a *ttParadox*, a *ttDBase* o *ttASCII*.Per tutti gli altri tipi di tabella, impostare *TableType* a *ttDefault*.
- Il nome della tabella che si vuole creare. Sia *TTable* che *TIBTable* hanno una proprietà *TableName* per il nome della nuova tabella. I dataset client non utilizzano un nome della tabella, ma, prima di salvare la nuova tabella, si dovrebbe specificare la proprietà *FileName*. Se si crea una tabella che duplica il nome di una tabella esistente, la tabella esistente e tutti i suoi dati saranno sovrascritti dalla tabella appena creata. La vecchia tabella e i suoi dati non potranno essere recuperati. Per evitare di sovrascrivere una tabella esistente, è possibile controllare la proprietà *Exists* in esecuzione. *Exists* è disponibile solo per *TTable* e *TIBTable*.
- I campi della nuova tabella. È possibile farlo in due modi:

- È possibile aggiungere definizioni di campo alla proprietà *FieldDefs*. In progettazione, fare doppio clic nell'Object Inspector sulla proprietà *FieldDefs* per richiamare il Collection editor. Utilizzare il Collection editor per aggiungere, cancellare o modificare le proprietà delle definizioni dei campi. In esecuzione, cancellare qualsiasi definizione di campo esistente e quindi utilizzare il metodo *AddFieldDef* per aggiungere ogni nuova definizione di campo. Per ogni nuova definizione di campo, impostare le proprietà dell'oggetto *TFieldDef* per specificare gli attributi desiderati per il campo.
- In alternativa, è possibile utilizzare anche componenti campo persistenti. In progettazione, fare doppio clic sul datset per richiamare il Fields editor. Nel Fields editor, fare clic destro e scegliere il comando New Field. Descrivere le proprietà di base del campo. Una volta creato il campo, è possibile modificarne le proprietà nell'Object Inspector selezionando il campo Fields Editor.

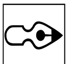
- Gli indici per la nuova tabella (facoltativo).In progettazione, fare doppio clic nell'Object Inspector sulla proprietà *IndexDefs* per richiamare il Collection editor. Utilizzare il Collection editor per aggiungere, cancellare o modificare le proprietà delle definizioni degli indici. In esecuzione, cancellare qualsiasi definizione di indice esistente e quindi utilizzare il metodo *AddIndexDef* per aggiungere ogni nuova definizione di indice. Per ogni nuova definizione di indice, impostare le proprietà dell'oggetto *TIndexDef* per specificare gli attributi desiderati per l'indice.



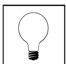
È impossibile definire indici per la nuova tabella se si utilizzano componenti campo persistenti invece di oggetti di definizione campo.

Per creare la tabella in progettazione, fare clic destro sul dataset e scegliere Create Table (*TTable*) o Create Data Set (*TClientDataSet*). Questo comando non viene visualizzato nel menu di scelta rapida finché non sono state specificate tutte le informazioni necessarie.

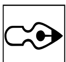
Per creare la tabella in esecuzione, chiamare il metodo *CreateTable* (*TTable* e *TIBTable*) o il metodo *CreateDataSet* (*TClientDataSet*).



Per creare la tabella è possibile impostare le definizioni in progettazione e poi chiamare il metodo *CreateTable* (o *CreateDataSet*) in esecuzione. Tuttavia, per poterlo fare è ncessario indicare che le definizioni specificate in progettazione devono essere salvate con il componente dataset. (per impostazione predefinita, le definizioni di campo e di indice vengono generate dinamicamente in esecuzione). Specificare che le definizioni devono essere salvate con il dataset impostando a **True** la proprietà *StoreDefs* del dataset.



Se si utilizza *TTable*, è possibile precaricare in progettazione le definizioni di campo e di indice di una tabella esistente. Impostare le proprietà *DatabaseName* e *TableName* per specificare la tabella esistente. Fare clic destro sul componente tabella e scegliere Update Table Definition. In questo modo, i valori delle proprietà *FieldDefs* e *IndexDefs* vengono impostati automaticamente per descrivere i campi e gli indici della tabella esistente. Dopo, reimpostare *DatabaseName* e *TableName* per specificare la tabella che si desidera creare, annullando qualsiasi richiesta di rinominare la tabella esistente.



Quando si creano tabelle Oracle8, è impossibile creare campi oggetto (campi ADT, campi array e campi dataset).

Il codice seguente crea una nuova tabella in fase di esecuzione e la associa all'alias BCDEMOS. Prima di creare la nuova tabella, verifica che il nome di tabella fornito non corrisponda al nome di una tabella esistente:

```
TTable *NewTable = new TTable(Form1);
NewTable->Active = false;
NewTable->DatabaseName = "BCDEMOS";
NewTable->TableName = Edit1->Text;
NewTable->TableType = ttDefault;
NewTable->FieldDefs->Clear();
TFieldDef *NewField = NewTable->FieldDefs->AddFieldDef(); // define first field
NewField->DataType = ftInteger;
NewField->Name = Edit2->Text;
NewField = NewTable->FieldDefs->AddFieldDef(); // define second field
NewField->DataType = ftString;
NewField->Size = StrToInt(Edit3->Text);
NewField->Name = Edit4->Text;
NewTable->IndexDefs->Clear();
TIndexDef *NewIndex = NewTable->IndexDefs->AddIndexDef(); // add an index
NewIndex->Name = "PrimaryIndex";
NewIndex->Fields = Edit2->Text;
NewIndex->Options << ixPrimary << ixUnique;
// Now check for prior existence of this table
bool CreateIt = (!NewTable->Exists);
if (!CreateIt)
   if (Application->MessageBox((AnsiString("Overwrite table ") + Edit1->Text +
                               AnsiString("?")).c_str(),
                               "Table Exists", MB_YESNO) == IDYES)
    CreateIt = true;
if (CreateIt)
  NewTable->CreateTable(); // create the table
```

Cancellazione di tabelle

TTable e *TIBTable* permettono di cancellare tabelle dalla tabella di database sottostante senza utilizzare SQL. Per cancellare una tabella in esecuzione, chiamare il metodo *DeleteTable* del dataset. Ad esempio, la seguente istruzione elimina la tabella sottostante un dataset:

```
CustomersTable->DeleteTable();
```



Quando si cancella una tabella con *DeleteTable*, la tabella e tutti i suoi dati vanno definitivamente perduti.

Se si utilizza *TTable*, è possibile anche cancellare le tabelle in progettazione: fare clic destro sul componente tabella e selezionare Delete Table dal menu di scelta rapida. La scelta di menu Delete Table è disponibile solo se il componente tabella rappresenta una tabella di database esistente (le proprietà *DatabaseName* e *TableName* specificano una tabella esistente).

Svuotamento di tabelle

Molti dataset di tipo tabella forniscono un singolo metodo che permette di cancellare tutte le righe di dati nella tabella.

- Per *TTable* e per *TIBTable*, è possibile cancellare tutti i record chiamando in esecuzione il metodo *EmptyTable*:

```
PhoneTable->EmptyTable();
```

- Per *TADOTable*, è possibile utilizzare il metodo *DeleteRecords*.

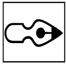
```
PhoneTable->DeleteRecords(arAll);
```

- Anche per *TSQLTable* è possibile utilizzare il metodo *DeleteRecords*. Si noti, tuttavia, che la versione *TSQLTable* di *DeleteRecords* non richiede mai alcun parametro.

```
PhoneTable->DeleteRecords();
```

- Per i dataset client, è possibile utilizzare il metodo *EmptyDataSet*.

```
PhoneTable->EmptyDataSet();
```



Per tabelle su server SQL, questi metodi vanno a buon fine solo se si ha il privilegio DELETE per la tabella.



Quando si svuota un dataset, i dati cancellati vanno definitivamente perduti.

Sincronizzazione di tabelle

Se si hanno due o più dataset che rappresentano la stessa tabella di database ma non condividono un componente datasource, ogni dataset ha la propria vista sui dati e il proprio record corrente. Se si accede ai record utilizzando ognuno dei due dataset, i record correnti dei componenti saranno diversi.

Se tutti i dataset sono istanze di *TTable* o di *TIBTable* o sono tutti dataset client, chiamando il metodo *GotoCurrent* è possibile forzare il record corrente di ognuno di questi dataset a eguagliare quello degli altri. *GotoCurrent* imposta il record corrente del proprio dataset al record corrente di un dataset corrispondente. Ad esempio, il seguente codice imposta il record corrente di *CustomerTableOne* in modo che sia lo stesso del record corrente di *CustomerTableTwo*:

```
CustomerTableOne->GotoCurrent(CustomerTableTwo);
```



ISe l'applicazione deve sincronizzare i dataset in questo modo, mettere i dataset in un modulo dati e includere l'header per il modulo dati in ciascuna unit che accede alle tabelle. Se occorre sincronizzare i dataset su schede distinte, bisogna includere il file header di quella scheda nella unit sorgente dell'altra scheda, e inoltre qualificare almeno uno dei dataset con il nome della relativa scheda. Per esempio:

```
CustomerTableOne->GotoCurrent(Form2->CustomerTableTwo);
```

Utilizzo di dataset di tipo query

Per utilizzare un dataset di tipo query,

1 Collocare il componente dataset appropriato in un modulo dati o su una scheda e impostarne la proprietà *Name* a un valore univoco, consono all'applicazione.

2 Identificare il server di database da interrogare. Ogni dataset di tipo query compie l'operazione in mdo differente, ma di solito si specifica un componente database *connection*:

- Per *TQuery*, specificare un componente *TDatabase* o un alias BDE utilizzando la proprietà *DatabaseName*.
- Per *TADOQuery*, specificare un componente *TADOConnection* utilizzando la proprietà *Connection*.
- Per *TSQLQuery*, specificare un componente *TSQLConnection* utilizzando la proprietà *SQLConnection*.
- Per *TIBQuery*, specificare un componente *TIBConnection* utilizzando la proprietà *Database*.

Per informazioni sull'utilizzo di componenti database connection, vedere Capitolo 21, "Connessione ai database".

3 Specificare un'istruzione *SQL* nella proprietà *SQL* del dataset e specificare in modo facoltativo qualsiasi parametro da passare all'istruzione. Per ulteriori informazioni, vedere "Specifica della query" a pagina 22-44 e "Utilizzo dei parametri nelle query" a pagina 22-46.

4 Se i dati della query devono essere utilizzati con controlli dati visuali, aggiungere al modulo dati un componente datasource e impostarne la proprietà *DataSet* al dataset di tipo query. Il componente datasource passa il risultato della query (detto anche *set* risultato) ai componenti data-aware affinché siano visualizzati. Connettere i componenti data-aware al datasource utilizzando le loro proprietà *DataSource* e *DataField*.

5 Attivare il componente query. Per query che restituiscono un *set* risultato, utilizzare la proprietà *Active* o il metodo *Open*. Per eseguire query che eseguono solo un'azione su una tabella e non restituiscono alcun set risultato, utilizza il metodo *ExecSQL* in esecuzione. Se si prevede di eseguire la query più di una volta, si potrebbe volere chiamare *Prepare* per inizializzare lo strato di accesso ai dati e assemblare nella query i valori dei parametri. Per informazioni sulla preparazione di una query, vedere "Preparazione di query" a pagina 22-50.

Specifica della query

Per dataset di tipo query, utilizzare la proprietà *SQL* per specificare l'istruzione *SQL* che dovrà eseguire il dataset.Alcuni dataset, come *TADODataSet*, *TSQLDataSet* e i dataset client, utilizzano una proprietà *CommandText* per svolgere lo stesso compito.

La maggior parte delle query che restituiscono record sono comandi SELECT. Di solito, esse definiscono i campi da includere, le tabelle da cui selezionare quei campi, le condizioni che limitano i record da includere e l'ordine del dataset risultante. Per esempio:

```
SELECT CustNo, OrderNo, SaleDate
FROM Orders
WHERE CustNo = 1225
ORDER BY SaleDate
```

Le query che non restituiscono record includono istruzioni di tipo DDL (Data Definition Language) o DML(Data Manipulation Language) diverse da istruzioni SELECT (ad esempio, i comandi INSERT, DELETE, UPDATE, CREATE INDEX e ALTER TABLE non restituiscono alcun record). Il linguaggio utilizzato nei comandi è specifico del server, ma di solito conforme allo standard SQL-92 per il linguaggio SQL.

Il comando SQL che si esegue deve essere conforme alle specifiche del server utilizzato. I dataset non sono in grado né di valutare ne di eseguire l'istruzione SQL. Essi si limitano solo a passare il comando al server affinché venga eseguito. Nella maggior parte dei casi, il comando SQL deve essere solo un'istruzione completa SQL, benché l'istruzione possa essere, se necessario, anche molto complessa (ad esempio, un'istruzione SELECT con una clausola WHERE che utilizza numerosi operatori logici annidati come AND e OR). Alcuni server supportano anche una sintassi "batch" che permette più istruzioni; se il server supporta tale sintassi, quando si specifica la query è possibile immettere più istruzioni.

Le istruzioni SQL utilizzate da query possono essere letterali o possono contenere parametri sostituibili. Query che utilizzano parametri sono chiamate query parametrizzate. Quando si utilizzano query parametrizzate, i valori effettivi assegnati ai parametri sono inseriti nella query prima di eseguire la query. L'utilizzo di query parametrizzate è molto flessibile, perché è possibile modificare al volo, in fase di esecuzione, la vista e l'accesso ai dati senza dovere modificare l'istruzione SQL. Per ulteriori informazioni sulle query parametrizzate, vedere "Utilizzo dei parametri nelle query" a pagina 22-46.

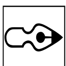
Specifica di una query utilizzando la proprietà *SQL*

Quando si utilizzando un vero dataset di tipo query (*TQuery*, *TADOQuery*, *TSQLQuery*, o *TIBQuery*), assegnare la query alla proprietà *SQL*. La proprietà SQL è un oggetto *TStrings*. Ogni singola stringa in questo oggetto *TStrings* è una riga distinta della query. L'uso di più righe non influisce sul modo in cui la query viene eseguita sul server, ma la modifica e il debug della query può risultare più semplice se l'istruzione viene suddivisa in più unità logiche:

```
MyQuery->Close();
MyQuery->SQL->Clear();
MyQuery->SQL->Add("SELECT CustNo, OrderNO, SaleDate");
MyQuery->SQL->Add("FROM Orders");
MyQuery->SQL->Add("ORDER BY SaleDate");
MyQuery->Open();
```

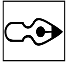
Il codice seguente mostra come modificare solo una singola riga in un'istruzione SQL esistente. In questo caso, la clausola ORDER BY esiste già sulla terza riga dell'istruzione. È referenziata mediante la proprietà *SQL* utilizzando un indice pari a 2.

```
MyQuery->SQL->Strings[2] = "ORDER BY OrderNO";
```



Quando si specifica o si modifica la proprietà *SQL*, il dataset deve essere chiuso.

In progettazione, per specificare la query utilizzare lo String List editor. Per visualizzare lo String List editor, fare clic nell'Object Inspector sul pulsante ellissi della proprietà *SQL*.



Con alcune versioni di C++Buider, se si utilizza *TQuery*, è possibile anche utilizzare SQL Builder per costruire una query in base a una rappresentazione visibile delle tabelle e dei campi in un database. Per utilizzare SQL Builder, selezionare il componente query, fare clic destro su esso per richiamare il menu di scelta rapida e scegliere Graphical Query Editor. Per apprendere l'utilizzo di SQL Builder, aprirlo e utilizzarne la Guida in linea.

Poiché la proprietà *SQL* è un oggetto *TStrings*, è possibile caricare il testo della query da un file chiamando il metodo *LoadFromFile* di *TStrings*:

```
MyQuery->SQL->LoadFromFile("custquery.sql");
```

Per copiare il contenuto di un oggetto string list nella proprietà *SQL* è possibile utilizzare anche il metodo *Assign* della proprietà *SQL*. Prima di copiare la nuova istruzione, il metodo *Assign* cancella automaticamente il contenuto corrente della proprietà *SQL*:

```
MyQuery->SQL->Assign(Memo1->Lines);
```

Specifica di una query utilizzando la proprietà CommandText

Quando si utilizza *TADODataSet*, *TSQLDataSet* o un dataset client, assegnare il testo dell'istruzione query alla proprietà *CommandText*:

```
MyQuery->CommandText = "SELECT CustName, Address FROM Customer";
```

In fase di progettazione, è possibile immettere la query direttamente nell'Object Inspector o, se il dataset ha già una connessione attiva al database, è possibile fare clic sul pulsante ellissi della proprietà *CommandText* per visualizzare il Command Text editor. Command Text editor elenca le tabelle disponibili e i campi in quelle tabelle, in modo da semplificare la composizione della query.

Utilizzo dei parametri nelle query

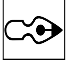
Un'istruzione SQL parametrizzata contiene parametri, o variabili, i cui valori possono essere modificati in progettazione o in esecuzione. I parametri possono sostituire valori di dati, come quelli utilizzati in una clausola WHERE per confronti, che appaiono in un'istruzione SQL. Comunemente, i parametri vengono utilizzati per i valori di dati passati all'istruzione. Ad esempio, nella seguente istruzione SELECT, i valori da inserire vengono passsati come parametri:

```
INSERT INTO Country (Name, Capital, Population)
VALUES (:Name, :Capital, :Population)
```

In questa istruzione SQL, *:Name*, *:Capital* e *:Population* :CustNumber sono dei segnaposti per i valori effettivi fornito dall'applicazione all'istruzione in fase di esecuzione. Notare che i nomi dei parametri iniziano con un due punti. Il due punti è necessario per poter distinguere i nomi dei parametri dai valori letterali. È possibile anche includere parametri senza nome aggiungendo alla query un punto interrogativo (?). I parametri senza nome vengono identificati in base alla posizione, dal momento che non hanno nomi univoci.

Prima che il dataset possa eseguire la query, è necessario fornire i valori a tutti i parametri presenti nel testo della query. *TQuery*, *TIBQuery*, *TSQLQuery* e i dataset

client utilizzano la proprietà *Params* per memorizzare questi valori. *TADOQuery* utilizza invece la proprietà *Parameters*. *Params* (o *Parameters*) è una raccolta di oggetti (*TParam*, o *TParameter*) in cui ogni oggetto rappresenta un singolo parametro. Quando si specifica il testo della query, il dataset genera questo set di oggetti parametro e (a seconda del tipo d dataset) inizializza tutte le rispettive proprietà che è in grado di desumere dalla query.



Impostando la proprietà *ParamCheck* a **false**, è possibile sopprimere la generazione automatica di oggetti parametro causata dalla modifica del testo della query. Questo è utile per istruzioni DDL (Data Definition Language) che contengono, come parte dell'istruzione DDL, parametri che non sono parametri della query. Ad esempio, l'istruzione DDL per creare una procedura registrata potrebbe definire parametri che fanno parte della procedura registrata. Se si imposta *ParamCheck* a **false**, si impedisce a questi parametri di essere scambiati per parametri della query.

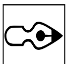
I valori dei parametri devono essere inglobati nell'istruzione SQL prima che venga eseguita per la prima volta. I componenti query lo fanno automaticamente anche se non si chiama esplicitamente il metodo *Prepare* prima di eseguire la query.



È una buona pratica di programmazione assegnare ai parametri nomi variabili, corrispondonti al nome effettivo della colonna a cui sono associati. Ad esempio, se un nome di colonna è "Number", allora il suo parametro corrispondente sarebbe ":Number". L'uso di nomi corrispondenti è particolarmente importante se il dataset utilizza un *datasource* per ottenere i valori dei parametri da un altro dataset. Questo procedimento è descritto in "Impostazione di relazioni master/detail mediante parametri" a pagina 22-49.

Fornitura dei parametri in progettazione

In progettazione, è possibile specificare i valori dei parametri utilizzando il Parameter collection editor. Per visualizzare il Parameter collection editor, fare clic nell'Object Inspector sul pulsante ellissi della proprietà *Params* o *Parameters*.Se l'istruzione SQL non contiene alcun parametro, nel Parameter collection editor non è elencato alcun oggetto.



Il Parameter collection editor è lo stesso editor di collection che viene visualizzato per altre proprietà di tipo collection. Dal momento che l'editor è condiviso con altre proprietà, il relativo menu contestuale contiene i comandi Add e *Delete*. Tuttavia, tali comandi non sono mai abilitati per i parametri delle query. L'unico modo per aggiungere o cancellare parametri è utilizzare l'istruzione SQL.

Per ciascun parametro, selezionarlo nel Parameter collection editor. Quindi utilizzare l'Object Inspector per modificarne le proprietà.

Quando si utilizzerà la proprietà *Params* (oggetti *TParam*), si vorrà esaminare o modificare quanto segue:

- La proprietà *DataType* elenca il tipo di dati per il valore del parametro. Per alcuni dataset, questo valore potrebbe già essere inizializzato correttamente. Se il dataset non è stato in grado di dedurre il tipo, *DataType* è *ftUnknown* ed è necessario modificarlo per indicare il tipo del valore del parametro.

  La proprietà *DataType* elenca il tipo di dati logico per il parametro. Di solito, questi tipi di dati si conformano ai tipi di dati del server. Per mappature specifiche dei

tipi di dati con i tipi di dati gestiti dal server, vedere la documentazione del meccanismo di accesso ai dati (BDE, dbExpress, InterBase).

- La proprietà *ParamType* elenca il tipo del parametro selezionato. Nel caso delle query, la proprietà è sempre *ptInput*, perché le query possono contenere solo parametri di *input*. Se il valore di *ParamType* è *ptUnknown*, modificarlo in *ptInput*.
- La proprietà *Value* specifica un valore per il parametro selezionato. Se l'applicazione fornisce il valore del parametro in esecuzione, è possibile non compilare *Value*.

Quando si utilizzerà la proprietà *Parameters* (oggetti *TParameter*), si vorrà esaminare o modificare quanto segue:

- La proprietà *DataType* elenca il tipo di dati per il valore del parametro. Per alcuni tipi di dati, è necessario fornire ulteriori informazioni:
  - La proprietà *NumericScale* indica il numero di cifre decimali per i parametri numerici.
  - La proprietà *Precision* indica il numero totale di cifre per i parametri numerici.
  - La proprietà *Size* indica il numero di caratteri nei parametri stringa.
- La proprietà *Direction* elenca il tipo del parametro selezionato. Nel caso delle query, la proprietà è sempre *pdInput*, perché le query possono contenere solo parametri di input.
- La proprietà *Attributes* controlla il tipo dei valori accettati dal parametro. *Attributes* può essere impostato a una combinazione di *psSigned*, *psNullable* e *psLong*.
- La proprietà *Value* specifica un valore per il parametro selezionato. Se l'applicazione fornisce il valore del parametro in esecuzione, è possibile non compilare *Value*.

Fornitura dei parametri in esecuzione

Per creare i parametri in esecuzione, è possibile utilizzare

- Il metodo *ParamByName* per assegnare i valori a un parametro in base al suo nome (non disponibile per *TADOQuery*)
- La proprietà *Params::Items* o *Parameters::Items* per assegnare i valori a un parametro in base alla posizione ordinale del parametro nell'istruzione SQL.
- La proprietà *Params::ParamValues* oppure *Parameters::ParamValues* per assegnare valori a uno o più parametri in una linea di comando singola, in base al nome di ciascun parametro.

Il codice seguente utilizza *ParamByName* per assegnare il testo di una casella di modifica al parametro *:Capital*:

```
SQLQuery1->ParamByName("Capital")->AsString = Edit1->Text;
```

Lo stesso codice può essere riscritto utilizzando la proprietà *Params* e un indice pari a 0 (si parte dall'assunto che il parametro *:Capital* sia il primo parametro nell'istruzione SQL):

```
SQLQuery1->Params->Items[0]->AsString = Edit1->Text;
```

Il comando sotto riportato imposta tre parametri per volta, usando la proprietà *Params::ParamValues*:

```
Query1->Params->ParamValues["Name;Capital;Continent"] =
  VarArrayOf(OPENARRAY(Variant, (Edit1->Text, Edit2->Text, Edit3->Text)));
```

Si noti che *ParamValues* utilizza dei Variants, evitando la necessità di effettuare la conversione di tipo dei valori.

Impostazione di relazioni master/detail mediante parametri

Per impostare una relazione master/detail in cui il set di dettaglio è un dataset di tipo query, è necessario specificare una query che utilizza parametri. Questi parametri fanno riferimento ai valori del campo corrente del dataset master. Perché i valori del campo corrente del dataset master cambiano dinamicamente in fase di esecuzione, è necessario collegare di nuovo i parametri del set di dettaglio ogni volta che cambia il record principale. Benché sia possibile scrivere del codice per compiere l'operazione utilizzando un gestore di evento, tutti i dataset di tipo query tranne *TIBQuery* forniscono un meccanismo più semplice grazie all'utilizzo della proprietà *DataSource*.

Se i valori dei parametri di una query parametrizzata non vengono collegati in fase di progettazione o specificati in esecuzione, i dataset di tipo query tentano di fornire un valore ai parametri in base alla proprietà *DataSource*. *DataSource* identifica un dataset differente che viene esaminato alla ricerca di nomi di campo che corrispondano ai nomi dei parametri non collegati. Questo dataset di ricerca può essere un qualsiasi tipo di dataset. Prima di creare il dataset di dettaglio che lo utilizza, è necessario creare e riempire il dataset di ricerca. Se nel dataset di ricerca vengono trovate le corrispondenze, il dataset di dettaglio collega i valori dei parametri ai valori dei campi nel record corrente a cui punta la sorgente dati.

Per illustrare il funzionamento, si prendano in esame due tabelle: una tabella di clienti e una tabella di ordini. Per ogni cliente, la tabella degli ordini contiene una serie di ordini fatti dal cliente. La tabella Customer include un campo ID che specifica un codice cliente univoco. La tabella degli ordini include un campo CustID che specifica l'ID del cliente che ha fatto un ordine.

Il primo passo consiste nel configurare il dataset Customer:

1. Aggiungere all'applicazione un dataset di tipo tabella e collegarlo alla tabella Customer.
2. Aggiungere un componente *TDataSource* di nome *CustomerSource*. Impostarne la proprietà *DataSet* al dataset aggiunto al punto 1. Questo datasource ora rappresenta il dataset Customer.
3. Aggiungere un dataset di tipo query e impostarne la proprietà SQL a

   ```
   SELECT CustID, OrderNo, SaleDate
   FROM Orders
   WHERE CustID = :ID
   ```

Si noti che il nome del parametro è identico al nome del campo nella tabella master (Customer).

4 Impostare la proprietà *DataSource* del dataset di dettaglio a *CustomerSource*. L'impostazione di questa proprietà rende il dataset di dettaglio una query collegata.

In esecuzione, al parametro *:ID* dell'istruzione SQL per il dataset di dettaglio non viene assegnato un valore, e pertanto il dataset cerca fra le colonne del dataset identificato da *CustomersSource* una corrispondenza del parametro in base al nome. *CustomersSource* ricava i propri dati dal dataset master, il quale, a sua volta, ricava i propri dati dalla tabella Customer. Poiché la tabella Customer contiene una colonna di nome "ID", il valore del campo ID nel record corrente del dataset master viene assegnato al parametro *:ID* dell'istruzione SQL del dataset di dettaglio. I dataset sono collegati tramite una relazione master-detail. Ogni volta che nel dataset *Customers* il record corrente cambia, viene eseguita l'istruzione SELECT del dataset di dettaglio per ricercare tutti gli ordini in base all'identificativo del cliente corrente.

Preparazione di query

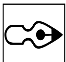
La preparazione di una query è un passo facoltativo che precede l'esecuzione della query. La preparazione di una query sottomette l'istruzione SQL e gli eventuali parametri allo strato di accesso ai dati e al server di database per l'analisi sintattica e per l'allocazione e l'ottimizzazione delle risorse. Con alcuni dataset, in fase di preparazione della query, il dataset può eseguire delle ulterori operazioni di configurazione. Queste operazioni migliorano le prestazioni della query, rendendo più rapida l'applicazione, specialmente quando funzionano con query aggiornabili.

Un'applicazione può preparare una query impostando la proprietà *Prepared* a **true**. Se non si prepara una query prima di eseguirla, il dataset la prepara automaticamente ogni volta che si chiamano i metodi *Open* o *ExecSQL*. Sebbene il dataset prepari le query automaticamente, è possibile migliorare le prestazioni preparando esplicitamente il dataset prima di aprirlo per la prima volta.

```
CustQuery->Prepared = true;
```

Quando si prepara esplicitamente il dataset, le risorse allocate per l'esecuzione dell'istruzione non vengono liberate finché non si imposta *Prepared* a **false**.

Impostare la proprietà *Prepared* a **false** se si desidera essere certi che il dataset venga preparato di nuovo prima di essere eseguito (ad esempio, nel caso si aggiunga un parametro).



Quando si modifica il testo della proprietà *SQL* di una query, il dataset si chiude automaticamente e annulla la preparazione della query.

Esecuzione di query che non restituiscono un set risultato

Quando una query restituisce un set di record (come una query SELECT), la si esegue esattamente come se si stesse riempiendo un qualsiasi dataset con dei record: si imposta *Active* a **true** o si chiama il metodo *Open*.

Comunque, spesso i comandi SQL non restituiscono alcun record. Tali comandi includono istruzioni di tipo DDL (Data Definition Language) o DML (Data Manipulation Language) diverse da istruzioni SELECT (ad esempio, i comandi INSERT, DELETE, UPDATE, CREATE INDEX e ALTER TABLE non restituiscono alcun record).

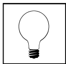
Per tutti i dataset di tipo query, è possibile eseguire una query che non restituisce un set risultato chiamando *ExecSQL*:

```
CustomerQuery->ExecSQL(); // Does not return a result set
```



Se la query viene eseguita più volte, è conveniente impostare la proprietà *Prepared* a **true**.

Benché la query non restituisca alcun record, si potrebbe voler conoscere il numero di record coinvolti nell'operazione (ad esempio, il numero di record cancellati da una query *DELETE*). La proprietà *RowsAffected* restituisce il numero di record coinvolti, dopo una chiamata a *ExecSQL*.



Quando in fase di progettazione non si sa se la query restituirà o meno un set risultato (ad esempio, se l'utente fornisce la query dinamicamente in fase di esecuzione), è possibile codificare entrambi i tipi di istruzioni di esecuzione della query in un blocco **try...catch**. Mettere una chiamata al metodo *Open* nella clausola **try**. Una query di azione viene eseguita quando si attiva la query con il metodo *Open*, ma in aggiunta si verifica un'eccezione. Controllare l'eccezione e ignorarla nel caso si limiti ad indicare la mancanza di un set risultato (ad esempio, *TQuery* indica un tale evento con un'eccezione di tipo *ENoResultSet*).

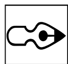
Utilizzo di set risultato unidirezionali

Se un dataset di tipo query restituisce un set risultato, riceve anche un cursor o un puntatore al primo record in quel set risultato. Il record puntato dal cursor è il record attualmente attivo.Il record corrente è quello i cui valori dei campi sono visualizzati in componenti data-aware associati al datasource del set risultato. A meno che non si stia utilizzando dbExpress, per impostazione predefinita, questo cursor è bidirezionale. Un cursor bidirezionale può spostarsi tra i suoi record sia in avanti che a ritroso. Il supporto di un cursor bidirezionale ha un peso gestionale aggiuntivo e può rallentare alcune query.

Se non è necessario essere in grado di spostarsi nel set risultato in entrambe le direzione, *TQuery* e *TIBQuery* permettono di migliorare le prestazioni della query richiedendo invece un cursor unidirezionale. Per richiedere un cursor unidirezionale, impostare la proprietà *UniDirectional* a **true**.

Impostare *UniDirectional* prima di preparare ed eseguire la query. Il seguente codice mostra l'impostazione di *UniDirectional* prima di preparare ed eseguire una query:

```
if (!CustomerQuery->Prepared)
{
  CustomerQuery->UniDirectional = true;
  CustomerQuery->Prepared = true;
}
CustomerQuery->Open(); // Returns a result set with a one-way cursor
```



Non confondere la proprietà *UniDirectional* con un dataset unidirezionale. I dataset unidirezionali (*TSQLDataSet*, *TSQLTable*, *TSQLQuery* e *TSQLStoredProc*) utilizzano dbExpress, che restituisce solo cursor unidirezionali. Oltre a limitare la capacità di spostamento a ritroso, i dataset unidirezionali non bufferizzano i record e pertanto hanno limitazioni aggiuntive (come l'incapacità di utilizzare i filtri).

Utilizzo di dataset di tipo procedura registrata

Il modo in cui l'applicazione utilizza una procedura registrata dipende da come è stata codificata la procedura registrata, dal fatto che restituisca o meno dati e in che modo, dallo specifico server di database utilizzato, o da una combinazione di tutti questi fattori.

In generale, per accedere a una procedura registrata su un server, un'applicazione deve:

1. Collocare il componente dataset appropriato in un modulo dati o su una scheda e impostarne la proprietà *Name* a un valore univoco, consono all'applicazione.
2. Identificare il server di database che definisce la procedura registrata. Ogni dataset di tipo procedura registrata lo fa in modo diverso, ma di solito si specifica un componente database connection:
   - Per *TStoredProc*, specificare un componente *TDatabase* o un alias BDE utilizzando la proprietà *DatabaseName*.
   - Per *TADOStoredProc*, specificare un componente *TADOConnection* utilizzando la proprietà *Connection*.
   - Per *TSQLStoredProc*, specificare un componente *TSQLConnection* utilizzando la proprietà *SQLConnection*.
   - Per *TIBStoredProc*, specificare un componente *TIBConnection* utilizzando la proprietà *Database*.

   Per informazioni sull'utilizzo di componenti database connection, vedere Capitolo 21, "Connessione ai database".
3. Specificare la procedura registrata da eseguire. Per la maggior parte dei dataset di tipo procedura registrata, lo si fa impostando la proprietà *StoredProcName*. Un'eccezione è *TADOStoredProc*, che ha invece una proprietà *ProcedureName*.
4. Se la procedura registrata restituisce un cursor da usare con controlli dati visuali, aggiungere al modulo dati un componente datasource e impostarne la proprietà *DataSet* al dataset di tipo procedura registrata. Connettere i componenti data-aware al datasource utilizzando le loro proprietà *DataSource* e *DataField*.
5. Fornire, se necessario, i valori dei parametri di input per la procedura registrata. Se il server non fornisce informazioni su tutti i parametri della procedura registrata, potrebbe essere necessario specificare ulteriori informazioni per i parametri di input, come i nomi dei parametri e i tipi di dati. Per i dettagli sui parametri delle procedure registrate, consultare "Operazioni con i parametri delle procedure registrate" a pagina 22-53.

6 Eseguire la procedura registrata. Per procedure registrate che restituiscono un cursor, utilizzare la proprietà *Active* o il metodo *Open*. Per eseguire procedure registrate che non restituisce alcun risultato o che restituiscono solo parametri di output, utilizzare il metodo *ExecProc* in fase di esecuzione. Se si prevede di eseguire la procedura registrata più di una volta, potrebbe essere oportuno chiamare *Prepare* per inizializzare lo strato di accesso ai dati e inglobare i valori dei parametri nella procedura registrata. Per informazioni sulla preparazione di una query, vedere "Esecuzione di procedure registrate che non restituiscono un set risultato" a pagina 22-56.

7 Elaborare gli eventuali risultati. Questi risultati possono essere restituiti come parametri risultato e parametri di output, oppure possono essere restituiti come set risultato che riempie il dataset di tipo procedura registrata. Alcune procedure registrate restituiscono più cursor. Per dettagli su come accedere ai cursor aggiuntivi, vedere "Prelievo di più set risultato" a pagina 22-57.

Operazioni con i parametri delle procedure registrate

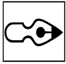
Esistono quattro tipi di parametri che è possibile associare a procedure registrate:

- *Parametri di input*, utilizzati per passare valori a una procedura registrata per l'elaborazione.
- *Parametri di output*, utilizzati da una procedura registrata per passare a un'applicazione i valori restituiti.
- *Parametri di input/output*, utilizzati per passare valori a una procedura registrata per l'elaborazione e utilizzati dalla procedura registrata per passare all'applicazione i valori restituiti.
- Un *parametro risultato*, utilizzato da alcune procedure registrate per restituire a un'applicazione un valore di errore o di stato. Una procedura registrata può restituire un solo parametro risultato.

Il fatto che una procedura registrata utilizzi un particolare tipo di parametro dipende sia dall'implementazione generale del linguaggio delle procedure registrate sul server di database sia dalla specifica istanza della procedura registrata. Per tutti i server, le singole procedure registrate possono utilizzare o meno parametri di input. D'altra parte, alcuni utilizzi dei parametri sono tipici del server. Ad esempio, su MS-SQL Server e Sybase, le procedure registrate restituiscono sempre un parametro risultato, mentre l'implementazione InterBase di una procedura registrata non restituisce mai un parametro risultato.

L'accesso ai parametri di una procedura registrata è garantito dalla proprietà *Params* (in *TStoredProc*, *TSQLStoredProc*, *TIBStoredProc*) o dalla proprietà *Parameters* (in *TADOStoredProc*). Quando si assegna un valore alla proprietà *StoredProcName* (o *ProcedureName*), il dataset genera automaticamente oggetti per ogni parametro della procedura registrata. Per alcuni dataset, se il nome della procedura registrata non vinespecificato fino al mometo dell'esecuzione, gli oggetti per ogni parametro devono essere creati da programma in quel preciso momento. La mancata specifica della procedura registrata e la creazione manualmente gli oggetti *TParam* o

TParameter permette di utilizzare un singolo dataset con tutte le procedure registrate disponibili.



Oltre ai parametri di output e risultato, alcune procedure registrate restituiscono un dataset. Le applicazioni possono visualizzare i record del dataset in controlli data-aware, ma devono elaborare separatamente i parametri di output e risultato.

Impostazione dei parametri in progettazione

È possibile specificare in fase progettazione i valori dei parametri della procedura registrata utilizzando il Parameter collection editor. Per visualizzare il Parameter collection editor, fare clic nell'Object Inspector sul pulsante ellissi della proprietà *Params* o *Parameters*.



È possibile assegnare valori ai parametri di input selezionandoli nel Parameter collection editor e utilizzando l'Object Inspector per impostarne la proprietà *Value*. Tuttavia, non si devono modificare i nomi o i tipi di dati dei parametri di input segnalati dal server. In caso contrario, quando si eseguirà la procedura registrata verrà sollevata un'eccezione.

Alcuni server non riportano i nomi dei parametri o i tipi di dati. In questi casi, è necessario configurare i parametri utilizzando manualmente l'editor *Parameter* collection. Per aggiungere parametri, fare clic destro e scegliere il comando Add. Per ogni parametro aggiunto, è necessario farne una descrizione completa. Anche se non è necessario aggiungere alcun parametro, si dovrebbero controllare le proprietà dei singoli oggetti parametro per assicurasi che siano corretti.

Se il dataset ha una proprietà *Params* (oggetti *TParam*), è necessario specificare correttamente le seguenti proprietà:

- La proprietà *Name* indica il nome del parametro così come definito dalla procedura registrata.
- La proprietà *DataType* riporta il tipo di dati per il valore del parametro. Quando si utilizza *TSQLStoredProc*, alcuni tipi di dati richiedono ulteriori informazioni:
  - La proprietà *NumericScale* indica il numero di cifre decimali per i parametri numerici.
  - La proprietà *Precision* indica il numero totale di cifre per i parametri numerici.
  - La proprietà *Size* indica il numero di caratteri nei parametri stringa.
- La proprietà *ParamType* riporta il tipo del parametro selezionato. Può essere *ptInput* (per parametri di input), *ptOutput* (per parametri di output), *ptInputOutput* (per parametri di input/output) o *ptResult* (per parametri risultato).
- La proprietà *Value* specifica un valore per il parametro selezionato. Non è mai possibile impostare i valori per i parametri di output e risultato. Questi tipi di parametri hanno valori che vengono impostati dall'esecuzione della procedura registrata. Nel caso dei parametri di input/output, se l'applicazione fornisce i valori dei parametri in esecuzione, è possibile non compilare *Value*.

Se il dataset utilizza una proprietà *Parameters* (oggetti *TParameter*), è necessario specificare correttamente le seguenti proprietà:

- La proprietà *Name* indica il nome del parametro così come definito dalla procedura registrata.
- La proprietà *DataType* riporta il tipo di dati per il valore del parametro. Per alcuni tipi di dati, è necessario fornire ulteriori informazioni:
  - La proprietà *NumericScale* indica il numero di cifre decimali per i parametri numerici.
  - La proprietà *Precision* indica il numero totale di cifre per i parametri numerici.
  - La proprietà *Size* il numero di caratteri nei parametri stringa.
- La proprietà *Direction* riporta il tipo del parametro selezionato. Può essere *pdInput* (per parametri di input), *pdOutput* (per parametri di output), *pdInputOutput* (per parametri di input/output) o *pdReturnValue* (per parametri risultato).
- La proprietà *Attributes* controlla il tipo dei valori accettati dal parametro. *Attributes* può essere impostato a una combinazione di *psSigned*, *psNullable* e *psLong*.
- La proprietà *Value* specifica un valore per il parametro selezionato. Non impostare i valori per i parametri di output e risultato. Nel caso dei parametri di input/output, se l'applicazione fornisce i valori dei parametri in esecuzione, è possibile non compilare *Value*.

Utilizzo di parametri in esecuzione

Con alcuni dataset, se il nome della procedura registrata non viene specificato fino al momento dell'esecuzione, per i parametri non verrà creato automaticamente alcun oggetto *TParam* ed dovranno essere creati da programma. È possibile fare ciò istanziando un nuovo oggetto *TParam* oppure il metodo *TParams::AddParam*:

```
TParam *P1, *P2;
StoredProc1->StoredProcName = "GET_EMP_PROJ";
StoredProc1->Params->Clear();
P1 = new TParam(StoredProc1->Params, ptInput);
P2 = new TParam(StoredProc1->Params, ptOutput);
try
{
  StoredProc1->Params->Items[0]->Name = "EMP_NO";
  StoredProc1->Params->Items[1]->Name = "PROJ_ID";
  StoredProc1->ParamByName("EMP_NO")->AsSmallInt = 52;
  StoredProc1->ExecProc();
  Edit1->Text = StoredProc1->ParamByName("PROJ_ID")->AsString;
}
__finally
{
  delete P1;
  delete P2;
}
```

Anche se non è necessario aggiungere in fase di esecuzione i singoli oggetti parametro, potrebbe essere necessario accedere ai singoli oggetti parametro per assegnare dei valori ai parametri di input e ottenere dei valori dai parametri di output. È possibile utilizzare il metodo *ParamByName* del dataset per accedere ai singoli parametri in base al loro nome. Ad esempio, il codice seguente imposta il

valore di un parametro di input/output, esegue la procedura registrata e recupera il valore restituito:

```
SQLDataSet1->ParamByName("IN_OUTVAR")->AsInteger = 103;
SQLDataSet1->ExecSQL();
int Result = SQLDataSet1->ParamByName("IN_OUTVAR")->AsInteger;
```

Preparazione delle procedure registrate

Come con i dataset di tipo query, i dataset di tipo procedura registrata devono essere preparati prima che possano eseguire la procedura registrata. La preparazione di una procedura registrata dice allo strato di accesso ai dati e al server di database di allocare le risorse per la procedura registrata e di inglobare i parametri. Queste operazioni possono migliorare prestazioni.

Se si tenta di eseguire una procedura registrata prima averla prepararla, sarà preparata automaticamente dal dataset, il quale, dopo l’esecuzione, provvederà ad annullarne la preparazione. Se si prevede di eseguire una procedura registrata più volte, risulta molto più efficiente prepararla esplicitamente impostando la proprietà *Prepared* a **true**.

```
MyProc->Prepared = true;
```

Quando si prepara esplicitamente il dataset, le risorse allocate per l’esecuzione dell’istruzione non vengono liberate finché non si imposta *Prepared* a **false**.

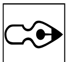
Impostare la proprietà *Prepared* a **false** se si desidera essere certi che il dataset venga preparato di nuovo prima dell’esecuzione (ad esempio, nel caso si cambiano i parametri utilizzando le procedure in overload di Oracle).

Esecuzione di procedure registrate che non restituiscono un set risultato

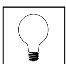
Quando una procedura registrata restituisce un cursor, la si esegue esattamente come se si stesse riempiendo un qualsiasi dataset con dei record: si imposta *Active* a **true** o si chiama il metodo *Open*.

Tuttavia, spesso le procedure registrate non restituiscono nessun dato, o restituiscono solo dei risultati nei parametri di output. È possibile eseguire una procedura registrata che non restituisce un set risultato chiamando *ExecProc*. Dopo avere eseguito la procedura registrata, è possibile utilizzare il metodo *ParamByName* per leggere il valore del parametro risultato o di un qualsiasi parametro di output:

```
MyStoredProcedure->ExecProc(); // Does not return a result set
Edit1->Text = MyStoredProcedure->ParamByName("OUTVAR")->AsString;
```



TADOStoredProc non ha un metodo *ParamByName*. Per ottenere i valori dei parametri di output quando di utilizza ADO, accedere agli oggetti parametro utilizzando la proprietà *Parameters*.



Se la procedure viene eseguita più volte, è conveniente impostare la proprietà *Prepared* a **true**.

Prelievo di più set risultato

Alcune procedure registrate restituiscono più *set* di record.Appena lo si apre, il dataset recupera solo il primo *set*. Se si utilizzano *TSQLStoredProc* oppure *TADOStoredProc*, è possibile accedere agli altri set di record chiamando il metodo *NextRecordSet* o *NextRecordset*:

```
TCustomSQLDataSet *DataSet2 = SQLStoredProc1->NextRecordSet();
```

Con *TSQLStoredProc*, il metodo *NextRecordSet* restituisce un nuovo componente *TCustomSQLDataSet* che fornisce l'accesso al successivo set di record. In *TADOStoredProc*, *NextRecordset* restituisce un'interfaccia che può essere assegnata alla proprietà *RecordSet* di un dataset ADO esistente. Per entrambe le classi, il metodo restituisce il numero di record nel dataset restituito come parametro di output.

La prima volta che si chiama *NextRecordSet*, il metodo restituisce il secondo set di record. La successiva chiamata a *NextRecordSet* restituisce nuovamente un terzo dataset e così via, finché non vi sono più altri set di record. Quando non vi sono più altri cursor, *NextRecordSet* restituisce NULL.

Capitolo

23

Operazioni con i componenti campo

Questo capitolo descrive le proprietà, gli eventi e i metodi comuni all'oggetto *TField* e ai suoi discendenti. I componenti campo rappresentano singoli campi (colonne) nei dataset. Questo capitolo descrive, inoltre, come usare i componenti campo per controllare la visualizzazione e la modifica dei dati nelle applicazioni.

I componenti campo sono sempre associati a un dataset. Non si utilizza mai un oggetto *TField* direttamente nelle applicazioni. Invece, ogni componente campo nell'applicazione è un discendente di *TField*, specifico per il tipo di dati di una colonna in un dataset. I componenti campo forniscono controlli data-aware, come *TDBEdit* e *TDBGrid*, per accedere ai dati in una particolare colonna del dataset associato.

Generalmente parlando, un singolo componente campo rappresenta le caratteristiche di una singola colonna, o campo di un dataset, come ad esempio il tipo di dato e la dimensione. Rappresenta anche le caratteristiche di visualizzazione dei campi, quali l'allineamento, il formato di visualizzazione e di modifica. Un componente *TFloatField*, ad esempio, ha quattro proprietà che influenzano direttamente l'aspetto dei suoi dati:

Tabella 23.1 Proprietà di TFloatField che influenzano la visualizzazione dei dati

Proprietà	Funzione
Alignment	Specifica se i dati sono visualizzati con allineamento a sinistra, a destra o al centro.
DisplayWidth	Specifica il numero di cifre da visualizzare contemporaneamente in un controllo.
DisplayFormat	Specifica il formato di visualizzazione dei dati (ad esempio, quante cifre decimali mostrare).
EditFormat	Specifica come visualizzare un valore in fase di modifica.

Man mano che si passa da un record a un altro all'interno del dataset, un componente campo permette di visualizzare e modificare il valore di quel campo nel record corrente.

I vari componenti campo hanno sia molte proprietà in comune (ad esempio *DisplayWidth* e *Alignment*), sia proprietà specifiche in base al relativo tipo di dati (ad esempio *Precision* per *TFloatField*). Ciascuna di queste proprietà influenza il modo in cui appaiono i dati in una scheda agli utenti di un'applicazione. Alcune proprietà, come *Precision*, possono anche influire sul valore che l'utente può immettere in un controllo durante la modifica o l'inserimento dei dati.

Tutti i componenti campo di un dataset sono dinamici (generati automaticamente in base alla struttura associata delle tabelle del database), o persistenti (generati in base ai nomi di campo e proprietà specifici impostati nel *Fields* editor). I campi dinamici e persistenti hanno potenza differente e sono adatti a tipi differenti di applicazioni. Le sezioni seguenti descrivono in maggior dettaglio i campi dinamici e persistenti e suggeriscono come scegliere fra l'uno e l'altro tipo.

Componenti campo dinamici

I componenti campo vengono generati dinamicamente per impostazione predefinita. Infatti, la prima volta che si pone un qualsiasi dataset in un modulo dati, si deve specificare in che modo quel dataset preleva i dati e aprirlo. Un componente campo è *dinamico* se viene creato automaticamente in base alle caratteristiche fisiche intrinseche dei dati rappresentati da un dataset. I dataset generano un componente campo per ciascuna colonna nei dati associati. L'esatto discendente *TField* creato per ciascuna colonna, viene determinato in base alle informazioni sul tipo di campo ricevute dal database o (per *TClientDataSet*) da un componente provider.

I campi dinamici sono temporanei. Essi esistono finché un dataset è aperto. Ogni volta che si riapre un dataset che usa campi dinamici, il dataset rigenera una serie completamente nuova di componenti campo dinamici in base alla struttura attiva dei dati del database associate al dataset. Se le colonne nei dati associati vengono modificate, la volta successiva in cui viene aperto un dataset che usa i componenti campo dinamici, anche i componenti campo generati automaticamente vengono modificati in conseguenza.

L'utilizzo dei campi dinamici è consigliato in applicazioni che richiedono flessibilità in fase di visualizzazione e di modifica dei dati. Ad esempio, per creare uno strumento di esplorazione del database come SQL explorer, si devono usare campi dinamici perché ogni tabella di database ha numero e tipi di colonne differenti. È bene usare i campi dinamici anche in applicazioni in cui l'interazione dell'utente con i dati si verifica soprattutto all'interno di componenti griglia e, come è noto, i dataset usati dall'applicazione subiscono frequenti modifiche.

Per usare campi dinamici in un'applicazione:

1. Posizionare i dataset e le sorgenti dati in un modulo dati.
2. Associare i dataset ai dati. Questo implica l'utilizzo di un componente connection o di un provider per collegarsi alla sorgente dati e l'impostazione di tutte le proprietà che specificano quali sono i dati che il dataset rappresenta.
3. Associare le sorgenti dati al dataset.

4 Collocare i controlli data-aware nelle schede dell'applicazione, aggiungere l'header del modulo dati nella unit di ciascuna scheda e associare ciascun controllo data-aware a un datasource nel modulo. Inoltre, associare un campo a ciascun controllo data-aware che lo richiede. Si noti che poiché si stanno utilizzando componenti campo dinamici, non c'è alcuna garanzia che il nome di campo che si specifica esisterà al momento dell'apertura del dataset.

5 Aprire il dataset.

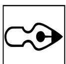
Indipendentemente dalla semplicità d'uso, i campi dinamici possono imporre delle limitazioni. Senza scrivere codice, non è possibile modificare le impostazioni predefinite di visualizzazione e modifica dei campi dinamici, non si può modificare, senza rischi, la sequenza con cui vengono visualizzati i campi dinamici e non si può impedire l'accesso ai campi del dataset. Non si possono creare campi aggiuntivi per il dataset, quali ad esempio campi calcolati o campi di consultazione, e non si può ridefinire il tipo di dato predefinito di un campo dinamico. Per aumentare il controllo e la flessibilità sui campi nelle applicazioni database, occorre richiamare Fields editor per creare componenti campo persistenti per il dataset.

Componenti campo persistenti

Per impostazione predefinita, i campi del dataset sono dinamici. Le loro proprietà e la disponibilità sono impostate automaticamente e non possono essere modificate in alcun modo. Per controllare appieno le proprietà e gli eventi di un campo, si devono creare per il dataset campi persistenti. I campi persistenti, consentono di:

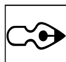
- Impostare o modificare la visibilità del campo o le caratteristiche in fase di progetto o in esecuzione.
- Creare nuovi campi, come campi di consultazione, campi calcolati e campi aggregati, che basano il proprio valore sui campi esistenti di un dataset.
- Eseguire la convalida dei dati immessi.
- Eliminare componenti campo dall'elenco dei componenti persistenti, per impedire all'applicazione di accedere a determinate colonne di un database associato.
- Definire nuovi campi solitamente per sostituire i campi esistenti, basati sulle colonne della tabella o della query associata al dataset.

In fase di progetto, è opportuno utilizzare Fields Editor per creare liste di componenti campo persistenti utilizzati dai dataset nell'applicazione. Le liste di campi persistenti sono registrate nell'applicazione e non cambiano anche in caso di variazione della struttura del database associato al dataset. Dopo aver creato i campi persistenti con Fields editor, è possibile creare anche i relativi gestori di evento che rispondono alle modifiche dei valori dei dati e convalidano i dati immessi.



Quando si creano campi persistenti per un dataset, in fase di progetto e in esecuzione sono disponibili per l'applicazione solo i campi selezionati. In fase di progetto, è sempre possibile utilizzare il Fields editor per aggiungere o rimuovere i campi persistenti di un dataset.

Tutti i campi usati da un singolo dataset sono o persistenti o dinamici. Non si possono mischiare questi due tipi di campo in un singolo dataset. Se per un dataset si creano campi persistenti e successivamente si vuole tornare ai campi dinamici, occorre rimuovere dal dataset tutti i campi persistenti. Per maggiori informazioni sui campi dinamici, consultare "Componenti campo dinamici" a pagina 23-2.



Una delle ragioni principali per usare i campi persistenti consiste nel fatto che essi consentono di controllare completamente l'aspetto e la visualizzazione dei dati. È anche possibile controllare l'aspetto delle colonne nelle griglie associate ai dati. Per informazioni su come controllare l'aspetto delle colonne nelle griglie consultare"Creazione di una griglia personalizzata" a pagina 19-18.

Creazione di campi persistenti

I componenti campo persistenti creati con Fields editor consentono di accedere ai dati associati tramite programma, in modo efficiente, leggibile e con salvaguardia del tipo. L'uso dei componenti campo persistenti, garantisce che ogni volta che si esegue un'applicazione, essa usa e visualizza sempre le stesse colonne, nello stesso ordine, anche se la struttura fisica del database associato è stata modificata.I componenti associati ai dati e il programma, che si aspettano di operare su determinati campi, funzionano sempre come previsto. Se si cancella o si modifica una colonna su cui è basato un componente campo persistente, C++Builder genera un'eccezione anziché eseguire l'applicazione e scontrarsi con una colonna inesistente o con dati che non coincidono con quelli previsti.

Per creare campi persistenti per un dataset:

1. Inserire un dataset in un modulo dati.
2. Collegare il dataset ai dati sottostanti. Di solito questo implica l'associazione del dataset a un componente connection o a un provider e la specifica di tutte le proprietà atte a descrivere i dati. Ad esempio, se si utilizza *TADODataSet*, è possibile impostare la proprietà *Connection* a un componente *TADOConnection* correttamente configurato e impostare la proprietà *CommandText* a una query valida.
3. Fare doppio clic sul componente dataset nel modulo dati per richiamare Fields editor. Questo possiede una barra di titolo, dei pulsanti di spostamento e una casella di riepilogo.

   La barra del titolo di Fields editor presenta sia il nome del modulo dati, o della scheda contenente il dataset, sia il nome del dataset stesso. Ad esempio, se viene aperto il dataset *Customers* nel modulo dati *CustomerData*, la barra del titolo visualizza la scritta 'CustomerData->Customers' o la parte di essa che può essere contenuta nella barra del titolo.

   Sotto la barra del titolo è presente una serie di pulsanti di navigazione che consentono, durante il progetto, di effettuare lo scorrimento, record per record, in un dataset attivo o di saltare al primo o all'ultimo record. Se il dataset non è attivo oppure è vuoto, i pulsanti di navigazione sono disattivati. Se il dataset è unidirezionale, i pulsanti per il passaggio all'ultimo record e al record precedente sono sempre inattivi.

La casella di riepilogo visualizza i nomi dei componenti campo persistenti per il dataset. Alla prima apertura di Fields editor per un nuovo dataset, l'elenco è vuoto perché i componenti campo del dataset sono dinamici e non persistenti. Se Fields editor viene richiamato per un dataset che possiede già componenti campo persistenti, i nomi dei campi saranno riportati nella casella di riepilogo.

4 Scegliere Add Fields dal menu contestuale del Fields editor.

5 Selezionare i campi da rendere persistenti nella finestra di dialogo Add Fields. Per impostazione predefinita, quando si apre la finestra di dialogo vengono selezionati tutti i campi. Tutti i campi selezionati dall'utente diventano campi persistenti.

La finestra di dialogo Add Fields verrà chiusa e i campi selezionati appariranno nella casella di riepilogo di Fields Editor. I campi riportati nella casella di riepilogo di Fields Editor sono persistenti. Se il dataset è attivo, i pulsanti di navigazione Next e (se il dataset non è unidirezionale) Last sopra la casella di riepilogo, sono abilitati.

Da questo momento in poi, a ogni apertura del dataset, non creerà più componenti campo dinamici per ogni colonna del database sottostante. Creerà invece solo componenti persistenti per i campi specificati.

Ad ogni apertura del dataset, verifica che ciascun campo persistente non calcolato esista o possa essere creato dai dati del database. Se ciò non è possibile, il dataset solleva un'eccezione avvertendo che il campo non è valido e il dataset non viene aperto.

Disposizione dei campi persistenti

L'ordine con cui sono elencati i componenti campo persistenti nella casella di riepilogo di Fields Editor è l'ordine predefinito con cui i campi appariranno in un componente griglia associato ai dati. L'ordine dei campi può essere modificato trascinando i campi nella casella di riepilogo.

Per cambiare la posizione di un campo:

1 Selezionare i campi. È possibile selezionare uno o più campi alla volta.

2 Trascinare i campi in una nuova posizione.

Se si seleziona una serie non contigua di campi e li si trascina in una nuova posizione, vengono inseriti come un blocco contiguo. Nell'ambito del blocco, l'ordine dei campi non viene modificato.

In alternativa, si può selezionare il campo e usare *Ctrl+Sù* e *Ctrl+Giù* per modificare l'ordine di un singolo campo dell'elenco.

Definizione di nuovi campi persistenti

Oltre a trasformare i campi esistenti del dataset in campi persistenti, è possibile creare anche campi persistente speciali in aggiunta o in sostituzione di altri campi persistenti di un dataset.

Nuovi campi persistenti servono solo ai fini della visualizzazione. I dati contenuti in esecuzione non vengono conservati, o perché esistono già altrove nel database o perché sono provvisori. La struttura fisica dei dati associati al dataset non viene modificata in alcun modo.

Per creare un nuovo componente campo persistente, occorre chiamare il menu contestuale di Fields editor e scegliere New Field. Appare così la finestra di dialogo New Field.

La finestra di dialogo New Fields contiene tre caselle di gruppo: Field properties, Field type e Lookup definition.

- La casella di gruppo Field properties permette di immettere informazioni generali sui componenti campo. Immettere il nome del campo nella casella di testo *Name*. Il nome immesso corrisponde alla proprietà *FieldName* del componente campo. La finestra di dialogo New Field utilizza questo nome per costruire un nome di componente nella casella di testo Component. Il nome visualizzato nella casella di testo Component corrisponde alla proprietà *Name* del componente campo ed è fornito solo a fini informativi (*Name* è l'identificativo utilizzato nel codice sorgente per fare riferimento al componente campo). La finestra di dialogo elimina qualsiasi cosa immessa direttamente nella casella di testo Component.
- La casella combinata Type del gruppo Field properties permette di specificare il tipo di dati del componente campo. È necessario indicare un tipo di dati per qualsiasi nuovo componente campo creato. Per esempio, per visualizzare valori di valuta in virgola mobile in un campo, selezionare *Currency* nell'elenco a discesa. Utilizzare la casella di testo Size per specificare il numero massimo di caratteri che possono essere visualizzati o immessi in un campo basato su stringhe, oppure la dimensione dei campi *Bytes* e *VarBytes*. Per tutti gli altri tipi di dati, Size è privo di significato.
- Il gruppo di pulsanti di opzione Field consente di specificare il tipo del nuovo componente campo da creare. Il tipo predefinito è Data. Se si sceglie Lookup, vengono attivate le caselle di testo Dataset e Source Fields nella casella del gruppo della definizione Lookup. È anche possibile creare campi di tipo Calculated e, se si sta lavorando con un dataset client, si possono creare campi di tipo InternalCalc o Aggregate. La tabella seguente descrive i tipi di campi che si possono creare:

Tabella 23.2 Tipi di campi persistenti particolari

Tipo di campo	Funzione
Data	Sostituisce un campo esistente (ad esempio per modificarne il tipo di dato)
Calculated	Visualizza i valori calcolati in esecuzione dal gestore di evento *OnCalcFields* di un dataset.
Lookup	Recupera valori da un determinato dataset in fase di esecuzione in base al criterio di ricerca specificato dall'utente. (Non supportato dai dataset unidirezionali)
InternalCalc	Visualizza i valori calcolati in esecuzione da un dataset client e memorizzati coi suoi dati.
Aggregate	Visualizza un valore di sintesi dei dati di un gruppo di record.

La casella di gruppo di definizione Lookup viene utilizzata solo per creare campi di ricerca. Per maggiori informazioni, consultare “Definizione di un campo di consultazione” a pagina 23-9.

Definizione di un campo dati

Un campo dati sostituisce un campo esistente in un dataset. Per esempio, per motivi di programmazione può essere necessario sostituire un *TSmallIntField* con un *TIntegerField*. Poiché il tipo di dati di un campo non può essere modificato direttamente, è necessario definire un nuovo campo per sostituirlo.



Anche se è un nuovo campo, il campo definito per sostituire un campo esistente deve derivare i valori dei propri dati da una colonna esistente in una tabella associata a un dataset.

Per creare un campo dati di sostituzione per un campo di una tabella sottostante un dataset:

1. Rimuovere il campo dall’elenco dei campi persistenti assegnati per il dataset, quindi scegliere New Field dal menu contestuale.
2. Nella finestra di dialogo New Field immettere nella casella di testo Name il nome di un campo esistente nella tabella del database. Non immettere un nuovo nome di campo. In questo modo, si specifica il nome del campo dal quale il nuovo campo deriverà i dati.
3. Scegliere un nuovo tipo di dati per il campo dalla casella combinata Type. Il tipo di dati scelto deve essere diverso da quello del campo che si sta sostituendo. Non è possibile sostituire un campo stringa di una determinata dimensione con un campo stringa di dimensione diversa. Si deve tenere presente che, anche se il tipo di dati sarà diverso, deve pur sempre essere compatibile con il tipo di dati del campo nella tabella sottostante.
4. All’occorrenza, immettere le dimensioni del campo nella casella di modifica Size. Size vale solo per i campi di tipo *TStringField*, *TBytesField* e *TVarBytesField*.
5. Selezionare Data nel gruppo Field type, se già non è selezionato.
6. Scegliere OK. La finestra di dialogo New Field verrà chiusa, il campo dati appena definito sostituirà il campo esistente specificato al punto 1 e la dichiarazione dei componenti nel modulo dati o nell’header della scheda sarà aggiornata.

Per modificare le proprietà o gli eventi associati a un componente campo, selezionare il nome del componente nella casella di riepilogo di Fields Editor, quindi modificare le relative proprietà o i relativi eventi con l’Object Inspector. Per ulteriori informazioni sulla modifica delle proprietà e degli eventi dei componenti campo, vedere “Impostazione di proprietà ed eventi dei campi persistenti” a pagina 23-11.

Definizione di un campo calcolato

Un campo calcolato visualizza i valori calcolati in esecuzione dal gestore di evento *OnCalcFields* di un dataset. Per esempio, è possibile creare un campo stringa che visualizza i valori concatenati di altri campi.

Per creare un campo calcolato nella finestra di dialogo New Field:

1 Nella casella di modifica Name, immettere un nome per il campo calcolato. Non immettere il nome di un campo esistente.

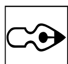
2 Nella casella combinata Type, scegliere un tipo di dati per il campo.

3 All'occorrenza, immettere le dimensioni del campo nella casella di modifica Size. Size vale solo per i campi di tipo *TStringField*, *TBytesField* e *TVarBytesField*.

4 Selezionare l'opzione Calculated o InternalCalc nel gruppo Field type. InternalCalc è disponibile solo se si opera con un dataset client. La differenza significativa fra questi tipi di campi calcolati è che i valori calcolati per un campo InternalCalc sono memorizzati e recuperati come parte dei dati del dataset client.

5 Scegliere OK. Il campo calcolato appena definito verrà automaticamente aggiunto alla fine dell'elenco dei campi persistenti nella casella di riepilogo di *Fields* Editor e la dichiarazione del componente verrà aggiunta automaticamente all'header della sheda o del modulo dati.

6 Inserire il codice che calcola i valori per il campo nel gestore di evento *OnCalcFields* del dataset. Per ulteriori informazioni sulla scrittura del codice per calcolare i valori dei campi, vedere "Programmazione di un campo calcolato" a pagina 23-8.



Per modificare le proprietà o gli eventi associati a un componente campo, selezionare il nome del componente nella casella di riepilogo di Fields Editor, quindi modificare le relative proprietà o i relativi eventi con l'Object Inspector. Per ulteriori informazioni sulla modifica delle proprietà e degli eventi dei componenti campo, vedere "Impostazione di proprietà ed eventi dei campi persistenti" a pagina 23-11.

Programmazione di un campo calcolato

Dopo aver definito un campo calcolato, è necessario scrivere il codice per calcolarne il valore. Altrimenti, il campo avrà sempre un valore nullo. Il codice per un campo calcolato viene inserito nell'evento *OnCalcFields* del relativo dataset.

Per programmare un valore per un campo calcolato:

1 Selezionare il componente dataset nell'elenco a discesa dell'Object Inspector.

2 Scegliere la pagina Object Inspector Events.

3 Fare doppio clic sulla proprietà *OnCalcFields* per richiamare o creare una procedura *CalcFields* per il componente dataset.

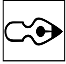
4 Scrivere il codice che imposta i valori e le altre proprietà del campo calcolato.

Per esempio, si supponga di aver creato un campo calcolato *CityStateZip* per la tabella *Customers* nel modulo dati *CustomerData*. *CityStateZip* dovrebbe visualizzare la città, lo stato e il codice di avviamento postale di un'azienda su una singola riga di un controllo associato ai dati.

Per aggiungere del codice alla procedura *CalcFields* per la tabella *Customers*, selezionare la tabella *Customers* dall'elenco a discesa dell'Object Inspector, accedere alla pagina Events e fare doppio clic sulla proprietà *OnCalcFields*.

La procedura *TCustomerData::CustomersCalcFields* appare nella finestra del codice sorgente della unit. Aggiungere il seguente codice alla procedura per calcolare il campo:

```
CustomersCityStateZip->Value = CustomersCity->Value + AnsiString(", ") +
    CustomersState->Value + AnsiString(" ") + CustomersZip->Value;
```

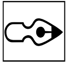


Quando si scrive il gestore di evento *OnCalcFields* per un campo calcolato internamente, è possibile migliorare le prestazioni controllando la proprietà *State* del dataset client e ricalcolando il valore solo quando *State* è *dsInternalCalc*. Per ulteriori informazioni consultare il paragrafo "Uso di campi calcolati internamente nei dataset client" a pagina 27-11.

Definizione di un campo di consultazione

Un campo di consultazione è un campo a sola lettura che, durante l'esecuzione, visualizza i valori in base ai criteri di ricerca specificati dall'utente. Nella sua forma più semplice, al campo di consultazione viene passato il nome di un campo esistente in cui eseguire la ricerca, un valore di campo da cercare e un campo diverso, in un dataset di consultazione, di cui dovrebbe visualizzare il valore.

Per esempio, si consideri un'applicazione per la gestione della vendita per corrispondenza che consenta a un operatore di utilizzare un campo di consultazione per determinare automaticamente la città e lo stato corrispondenti a un codice di avviamento postale indicato da un cliente. La colonna in cui eseguire la ricerca potrebbe essere denominata *ZipTable->Zip*, il valore da cercare è il codice di avviamento postale del cliente in *Order->CustZip* e i valori da restituire sarebbero quelli per le colonne *ZipTable->City* e *ZipTable->State* per il record in cui *ZipTable->Zip* corrisponde al valore attivo nel campo *Order->CustZip*.



I dataset unidirezionali non supportano i campi di consultazione.

Per creare un campo di consultazione nella finestra di dialogo New Field:

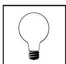
1. Nella casella di testo Name, immettere un nome per il campo di consultazione. Non immettere il nome di un campo esistente.
2. Nella casella combinata Type, scegliere un tipo di dati per il campo.
3. All'occorrenza, immettere le dimensioni del campo nella casella di modifica *Size*. *Size* vale solo per i campi di tipo *TStringField*, *TBytesField* e *TVarBytesField*.
4. Selezionare Lookup nel gruppo Field type. Selezionando Lookup, vengono abilitate le caselle combinate Dataset e Key Fields.
5. Scegliere nell'elenco a discesa della casella combinata Dataset il dataset in cui cercare i valori di campi. Il dataset di consultazione deve essere diverso dal dataset del componente campo, altrimenti in fase di esecuzione verrà sollevata un'eccezione dovuta a un riferimento circolare. Specificando un dataset di consultazione, le caselle combinate Lookup Keys e Result Field vengono abilitate.
6. Scegliere nell'elenco a discesa Key Fields un campo del dataset corrente per cui devono essere trovati dei valori corrispondenti. Per trovare corrispondenze per più di un campo, invece di scegliere i nomi dei campi dall'elenco a discesa, immetterli direttamente. Separare più nomi di campo con punti e virgola. Se si usa più di un campo, usare componenti campo persistenti.
7. Scegliere nell'elenco a discesa Lookup Keys un campo del dataset di consultazione da confrontare con il campo Source Fields specificato al punto 6. Se si specifica più

di un campo chiave, specificare lo stesso numero di chiavi di ricerca. Per specificare più di un campo, immettere direttamente i nomi dei campi, separandoli con dei punti e virgola.

8 Scegliere nell'elenco a discesa Result Field un campo del dataset di consultazione da restituire come valore del campo di consultazione che si sta creando.

Quando si progetta e si esegue un'applicazione, i valori del campo di consultazione vengono determinati prima di calcolare i valori dei campi calcolati. È possibile basare i campi calcolati sui campi di consultazione, ma non viceversa.

È possibile utilizzare la proprietà *LookupCache* per affinare il modo in cui vengono stabiliti i campi di consultazione. *LookupCache* determina se i valori di un campo di consultazione devono essere caricati in memoria appena si apre il dataset, o se devono essere consultati dinamicamente ogni volta che il record attivo del dataset viene modificato. *LookupCache* determina se i valori di un campo di consultazione devono essere caricati in memoria appena si apre il dataset, o se devono essere consultati dinamicamente ogni volta che il record attivo del dataset viene modificato. Conviene impostare *LookupCache* a **true** per mettere in cache i valori di un campo di consultazione quando è improbabile che il *LookupDataSet* venga modificato e il numero dei singoli valori di consultazione è piccolo. Mettere in cache i valori di consultazione può incrementare le prestazioni, perché i valori di consultazione di ogni serie di valori *LookupKeyFields* vengono precaricati appena si apre il *DataSet*. Quando si modifica il record attivo del *DataSet*, invece di accedere a *LookupDataSet*, l'oggetto campo può localizzare il suo *Value* nella cache. Questo incremento di prestazioni è particolarmente significativo se il *LookupDataSet* si trova su una rete che ha un accesso lento.



È possibile usare una cache di ricerca per fornire i valori tramite programma anziché tramite un dataset secondario. Assicurarsi che la proprietà *LookupDataSet* sia NULL. Quindi, utilizzare il metodo *Add* della proprietà *LookupList* per riempirlo con i valori di consultazione. Impostare la proprietà *LookupCache* a **true**. Il campo userà l'elenco di ricerca fornito senza sovrascriverlo con i valori di un dataset di ricerca.

Se i *KeyFields* di ogni record di *DataSet* hanno valori diversi, il carico di lavoro derivante dal cercare i valori nella cache può essere maggiore di qualsiasi beneficio in prestazioni fornito dalla cache. Il carico di lavoro derivante dal cercare i valori nella cache aumenta col numero di valori distinti che possono essere richiesti da *KeyFields*.

Se *LookupDataSet* è volatile, mettere in cache valori di consultazione può condurre a risultati imprecisi. Per aggiornare i valori nella cache di consultazione occorre chiamare *RefreshLookupList*. *RefreshLookupList* rigenera la proprietà *LookupList* che contiene il valore del *LookupResultField* per ogni serie di valori di *LookupKeyFields*.

Quando in l'esecuzione si imposta *LookupCache*, per inizializzare la cache occorre chiamare *RefreshLookupList*.

Definizione di un campo di aggregazione

Un campo di aggregazione visualizza valori da un'aggregazione di mantenimento in un dataset cliente. Un'aggregazione è un calcolo che riepiloga i dati di un gruppo di record. Per informazioni più dettagliate sulle aggregazioni di manutenzione, consultare "Uso delle aggregazioni di manutenzione" a pagina 27-12.

Per creare un campo di aggregazione nella finestra di dialogo NewField:

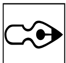
1 Immettere un nome per il campo nella casella di modifica Name. Non immettere il nome di un campo esistente.
2 Scegliere il tipo di dati di aggregazione per il campo dalla casella combinata Type.
3 Selezionare Aggregate nel gruppo di pulsanti di opzioni per Field type.
4 Scegliere OK. Il campo di aggregazione appena definito viene aggiunto automaticamente al dataset client, e la sua proprietà *Aggregates* viene poi aggiornata per includere la specifica di aggregazione appropriata.
5 Inserire l'operazione di calcolo prevista per l'aggregazione nella proprietà *ExprText* del campo di aggregazione appena creato. Per ulteriori informazioni sulla definizione di un campo di aggregazione, consultare il paragrafo "Specifica delle aggregazioni" a pagina 27-12.

Una volta creato un *TAggregateField* persistente, è possibile collegare un controllo *TDBText* al campo di aggregazione. Il controllo *TDBText* visualizzerà poi il valore del campo di aggregazione che è pertinente al record corrente del dataset client sottostante.

Cancellazione di componenti campo persistenti

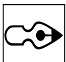
La cancellazione di un componente campo persistente è utile per accedere a un sottoinsieme di colonne disponibili in una tabella e per definire un proprio campo persistente per sostituire una colonna di una tabella. Per rimuovere uno o più componenti di campo persistenti di un dataset occorre:

1 Selezionare il campo o i campi da rimuovere nella casella di riepilogo Fields editor.
2 Premere *Canc*.



I campi selezionati possono essere cancellati anche chiamando il menu contestuale e scegliendo Delete.

I campi rimossi non sono più disponibili per il dataset e non possono essere visualizzati da controlli associati ai dati. È sempre possibile ricreare un componente campo persistente cancellato per errore, ma qualsiasi modifica apportata in precedenza alle proprietà o agli eventi si perde. Per maggiori informazioni, consultare "Creazione di campi persistenti" a pagina 23-4.



Se si rimuovono tutti i componenti campo persistenti per un dataset, questo torna a usare i componenti campo dinamici per ogni colonna della tabella sottostante del database.

Impostazione di proprietà ed eventi dei campi persistenti

In fase di progetto è possibile impostare proprietà e personalizzare eventi per i componenti campo persistenti. Le proprietà controllano il modo in cui un campo viene visualizzato da un componente associato ai dati, determinando per esempio se

può apparire in *TDBGrid* o se il suo valore può essere modificato. Gli eventi controllano quello che accade quando i dati di un campo vengono recuperati, modificati, impostati o convalidati.

Per impostare le proprietà di un componente campo o per scrivere gestori di evento personalizzati per esso, selezionare il componente nel Fields editor o nell'elenco dei componenti dell'Object Inspector.

Impostazione delle proprietà di visualizzazione e di modifica in progettazione

Per modificare le proprietà di visualizzazione di un componente campo selezionato, passare sulla pagina Properties della finestra dll'Object Inspector. La seguente tabella riepiloga le proprietà di visualizzazione che è possibile modificare.

Tabella 23.3 Proprietà dei componenti campo

Proprietà	Funzione
Alignment	Allinea a sinistra, allinea a destra o centra il contenuto di un campo in un componente associato ai dati.
ConstraintErrorMessage	Specifica il testo da visualizzare quando le modifiche contrastano con una condizione di vincolo.
CustomConstraint	Specifica un vincolo locale da applicare ai dati durante la modifica.
Currency	Solo campi numerici. **true**: visualizza valori monetari. **false** (impostazione predefinita): non visualizza valori monetari.
DisplayFormat	Specifica il formato dei dati visualizzati in un componente associato ai dati.
DisplayLabel	Specifica il nome di colonna di un campo in un componente griglia associato ai dati.
DisplayWidth	Specifica la larghezza, in caratteri, di una colonna di griglia che visualizza questo campo.
EditFormat	Specifica il formato di modifica dei dati in un componente associato ai dati.
EditMask	In un campo modificabile, limita l'immissione di dati ai soli tipi e intervalli di caratteri specificati e indica i caratteri speciali non modificabili che appaiono nel campo (trattini, parentesi e così via).
FieldKind	Specifica il tipo di campo da creare.
FieldName	Specifica il nome effettivo di una colonna di tabella da cui il campo trae il proprio valore e tipo di dati.
HasConstraints	Indica se ci sono condizioni di vincolo imposte su un campo.
ImportedConstraint	Specifica un vincolo SQL importato dal Data Dictionary o da un server SQL.
Index	Specifica l'ordine dei campi in un dataset.
LookupDataSet	Specifica la tabella usata per consultare i valori del campo quando *Lookup* è impostato a **true**.
LookupKeyFields	Specifica il campo o i campi del dataset di consultazione che devono corrispondere quando si effettua una consultazione.
LookupResultField	Specifica il campo del dataset di consultazione da cui copiare i valori da inserire in questo campo.
MaxValue	Solo campi numerici. Specifica il valore massimo che un utente può immettere nel campo.

Tabella 23.3 Proprietà dei componenti campo

Proprietà	Funzione
MinValue	Solo campi numerici. Specifica il valore minimo che un utente può immettere nel campo.
Name	Specifica il nome del componente campo in C++Builder.
Origin	Specifica il nome del campo come appare nel database associato.
Precision	Solo campi numerici. Specifica il numero di cifre significative.
ReadOnly	**true**: visualizza i valori dei campi nei controlli data-aware, ma ne impedisce la modifica. **false** (impostazione predefinita): consente la visualizzazione e la modifica dei valori dei campi.
Size	Specifica il numero massimo di caratteri che possono essere visualizzati o immessi in un campo di tipo stringa, oppure la dimensione in byte dei campi *TBytesField* e *TVarBytesField*.
Tag	Zona di memoria per interi lunghi, a disposizione dei programmatori in ogni componente per l'utilizzo secondo le esigenze.
Transliterate	**true** (impostazione predefinita): specifica che la traduzione da e verso le rispettive impostazioni locali si verificherà mentre i dati vengono trasferiti dal dataset al database. **false**: La traduzione relativa alle impostazioni locali non si verifica.
Visible	**true** (impostazione predefinita): consente la visualizzazione di un campo in una griglia data-aware. **false**: Impedisce la visualizzazione di un campo in un componente griglia associato ai dati. I componenti definiti dall'utente possono prendere decisioni relative alla visualizzazione in base a questa proprietà.

Non tutte le proprietà sono disponibili per tutti i componenti campo. Per esempio, un componente campo di tipo *TStringField* non possiede proprietà *Currency*, *MaxValue* o *DisplayFormat*, mentre un componente di tipo *TFloatField* non possiede la proprietà *Size*.

Anche se lo scopo della maggior parte delle proprietà è semplice, alcune proprietà, quali *Calculated*, richiedono ulteriori operazioni di programmazione perché siano veramente utili. Altre, quali *DisplayFormat*, *EditFormat* e *EditMask*, sono correlate; le loro impostazioni devono essere coordinate. Per ulteriori informazioni sull'uso di *DisplayFormat*, *EditFormat* ed *EditMask*, vedere "Controllo e mascheratura dell'input" a pagina 23-15.

Impostazione delle proprietà del componente campo in esecuzione

In fase di esecuzione è possibile usare e gestire le proprietà del componente campo. Accedere ai componenti campo persistenti in base al nome, dove il nome può essere ottenuto concatenando il nome del campo al nome del dataset.

Il codice che segue, ad esempio, imposta a **true** la proprietà *ReadOnly* per il campo *CityStateZip* nella tabella *Customer*:

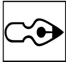
```
CustomersCityStateZip->ReadOnly = true;
```

E questa istruzione modifica l'ordine dei campi impostando a 3 la proprietà *Index* del campo *CityStateZip* nella tabella *Customer*:

```
CustomersCityStateZip->Index = 3;
```

Creazione di set di attributi per i componenti campo

Quando vari campi dei dataset utilizzati dall'applicazione condividono le stesse proprietà di formattazione (quali *Alignment*, *DisplayWidth*, *DisplayFormat*, *EditFormat*, *MaxValue*, *MinValue*, e così via), è più pratico impostare le proprietà per un solo campo e memorizzare quindi queste proprietà come set di attributi nel Data Dictionary. I set di attributi memorizzati nel dizionario dati possono essere facilmente applicati ad altri campi.



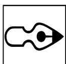
Set di attributi e Data Dictionary sono solo disponibili per i dataset BDE.

Per creare un set di attributi basato su un componente campo in un dataset:

1. Fare doppio clic sul dataset per richiamare il Fields editor.
2. Selezionare il campo per cui impostare le proprietà.
3. Impostare le proprietà desiderate per il campo nell'Object Inspector.
4. Fare clic con il pulsante destro del mouse sulla casella di riepilogo di Fields Editor per aprire il menu contestuale.
5. Scegliere Save Attributes per salvare le impostazioni delle proprietà del campo attivo come set di attributi nel Data Dictionary.

Il nome predefinito del set di attributi è quello del campo attivo. Se dal menu contestuale si sceglie Save Attributes As, invece di Save Attributes, è possibile specificare per il set di attributi un nome diverso.

Una volta che si è creato un nuovo set di attributi e lo si è aggiunto al Data Dictionary, è possibile poi associarlo ad altri componenti campo persistenti. Anche se in seguito si rimuoverà l'associazione, il set di attributi resta definito nel Data Dictionary.



È inoltre possibile creare set di attributi direttamente da SQL Explorer. Quando si crea un set di attributi utilizzando SQL Explorer, tale set viene aggiunto al Data Dictionary, ma non viene applicato a nessun campo. SQL Explorer consente di specificare altri due attributi: un tipo di campo (*TFloatField*, *TStringField*, e così via) e un controllo data-aware (come *TDBEdit*, *TDBCheckBox*, e così via) che vengono automaticamente collocati su una scheda non appena si trascina sulla scheda un campo basato sull'attributo impostato. Per ulteriori informazioni, vedere la guida in linea relativa a SQL Explorer.

Associazione di set di attributi ai componenti campo

Quando vari campi dei dataset utilizzati dall'applicazione condividono le stesse proprietà di formattazione (quali *Alignment*, *DisplayWidth*, *DisplayFormat*, *EditFormat*, *MaxValue*, *MinValue*, e così via), e queste impostazioni di proprietà sono state salvate come set di attributi nel Data Dictionary, questi set di attributi possono essere facilmente applicati ai campi, senza dover ricreare manualmente le impostazioni per ciascun campo. Inoltre, se in seguito vengono modificate nel Data Dictionary le impostazioni degli attributi, le modifiche verranno applicate automaticamente a tutti i campi associati al set non appena verranno aggiunti al dataset nuovi componenti campo.

Per applicare un set di attributi a un componente campo

1 Fare doppio clic sul dataset per richiamare il Fields editor.
2 Selezionare il campo cui applicare un set di attributi.
3 Chiamare il menu contestuale e scegliere Associate attributes.
4 Nella finestra di dialogo Attribute set name, selezionare o immettere Associate attributes da applicare. Se nel Data Dictionary c'è un set di attributi che ha lo stesso nome del campo attivo, il nome di quel set di attributi apparirà nella casella di testo.



Se in una fase successiva si modifica un set di attributi presente nel Data Dictionary, è necessario applicare nuovamente il set di attributi a ciascun componente campo che ne fa uso. Quando si riapplicano gli attributi è possibile richiamare il Fields editor e selezionare più componenti campo all'interno di un dataset.

Rimozione delle associazioni degli attributi

Se si cambia idea sull'associazione di un set di attributi a un campo, è possibile rimuovere l'associazione seguendo questi passi:

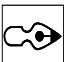
1 Chiamare il Fields editor per il dataset che contiene il campo.
2 Selezionare il campo o i campi da cui rimuovere l'associazione dell'attributo.
3 Chiamare il menu contestuale di Fields editor e scegliere Unassociate Attributes.



Annullando l'associazione di un attributo impostato non si modifica alcuna proprietà del campo. Il campo conserva le impostazioni che aveva quando gli è stato applicato il set di attributi. Per modificare queste proprietà, selezionare il campo in Fields editor e impostarne le proprietà con l'Object Inspector.

Controllo e mascheratura dell'input

La proprietà *EditMask* consente di controllare il tipo e l'intervallo dei valori che un utente può immettere in un componente data-aware associato a componenti *TStringField*, *TDateField*, *TTimeField*, *TDateTimeField*, e *TSQLTimeStampField*. È possibile usare maschere esistenti o crearne di proprie. Il modo più semplice per utilizzare e creare maschere di inserimento dati è usare Input Mask editor. È possibile, comunque, immettere le maschere direttamente nel campo *EditMask* nell'Object Inspector.



Per componenti *TStringField*, la proprietà *EditMask* rappresenta anche il formato di visualizzazione.

Per richiamare l'Input Mask editor per un componente campo:

1 Selezionare il componente in Fields Editor o nell'Object Inspector.
2 Fare clic sulla pagina Properties nell'Object Inspector.
3 Fare doppio clic sulla colonna dei valori del campo EditMask nell'Object Inspector, oppure fare clic sul pulsante con i puntini di sospensione. Verrà richiamato Input Mask editor.

La casella di testo Input Mask consente di creare e modificare un formato di maschera. La griglia Sample Masks consente di selezionare una maschera predefinita. Se si seleziona una maschera di esempio, il relativo formato appare nella casella di testo Input Mask, in cui è possibile modificarlo o utilizzarlo senza variazioni. Nella casella di testo Test Input è possibile collaudare le immissioni consentite all'utente per le maschere.

Il pulsante Masks consente di caricare nella griglia Sample Masks un set di maschere personalizzate – se ne sono state create – in modo da semplificare la selezione.

Uso dei formati predefiniti per i campi numerici, data e ora

C++Builder possiede routine incorporate per i formati di visualizzazione e modifica e una formattazione predefinita intelligente per i componenti *TFloatField*, *TCurrencyField*, *TIntegerField*, *TSmallIntField*, *TWordField*, *TDateField*, TdateTimeField e *TTimeField*. Per utilizzare queste routine, non occorre fare nulla.

La formattazione predefinita viene eseguita dalle seguenti routine:

Tabella 23.4 Routine di formattazione dei componenti campo

Routine	Utilizzata da...
FormatFloat	*TFloatField*, *TCurrencyField*
FormatDateTime	*TDateField*, *TTimeField*, *TDateTimeField,*
SQLTimeStampToString	*TSQLTimeStampField*
FormatCurr	*TCurrencyField, TBCDField*
BcdToStrF	*TFMTBCDField*

Per un determinato componente campo sono disponibili solo le proprietà di formato appropriate al proprio tipo di dati.

Le convenzioni di formattazione predefinita per i valori di data, ora, valuta e per i valori numerici si basano sulle proprietà Impostazioni internazionali del Pannello di controllo. Ad esempio, utilizzando le impostazioni predefinite per gli Stati Uniti, una colonna *TFloatField* con la proprietà *Currency* impostata a **true** imposta la proprietà *DisplayFormat* per il valore 1234.56 a $1234.56, mentre *EditFormat* è 1234.56.

In fase di progetto o durante l'esecuzione, è possibile modificare le proprietà *DisplayFormat* e *EditFormat* di un componente campo per ridefinire le impostazioni di visualizzazione predefinite per quel campo. Si possono scrivere anche i gestori di evento *OnGetText* e *OnSetText* per compiere una formattazione personalizzata dei componenti campo in fase di esecuzione.

Gestione degli eventi

Come molti componenti, i componenti campo hanno eventi associati. È possibile assegnare metodi che svolgano il compito di gestori di questi eventi. Grazie alla scrittura di questi eventi, è possibile reagire al verificarsi di questi eventi che riguardano l'immissione dei dati nei campi tramite controlli associati ai dati e

compiere operazioni specifiche. La seguente tabella elenca gli eventi associati ai componenti campo:

Tabella 23.5 Eventi dei componenti campo

Evento	Funzione
OnChange	Chiamato quando il valore di un campo cambia.
OnGetText	Chiamato quando si recupera il valore di un componente campo per visualizzarlo o modificarlo.
OnSetText	Chiamato quando si imposta il valore di un componente campo.
OnValidate	Chiamato per convalidare il valore di un componente campo ogni volta che viene modificato in seguito a un'operazione di modifica o di inserimento.

Gli eventi *OnGetText* e *OnSetText* sono utili principalmente per i programmatori che desiderano eseguire formattazioni personalizzate che esulano dalle funzioni di formattazione incorporate. *OnChange* è utile per eseguire operazioni specifiche dell'applicazione associate alla modifica dei dati, quali l'abilitazione o la disabilitazione di menu o di controlli visuali. *OnValidate* è utile per controllare nell'applicazione la convalida dei dati immessi prima di restituire i valori a un server di database.

Per scrivere un gestore di evento per un componente campo:

1. Selezionare il componente.
2. Selezionare nell'Object Inspector la pagina Events.
3. Fare doppio clic sul campo Value relativo al gestore di evento, per visualizzarne la finestra del codice sorgente.
4. Creare o modificare il codice del gestore.

Operazioni con i metodi del componente campo in esecuzione

I metodi dei componenti campo disponibili in esecuzione consentono di convertire i valori del campo da un tipo di dati a un altro e di impostare il fuoco sul primo controllo associato ai dati in una scheda associata con un componente di campo.

Il controllo del fuoco nei componenti associati ai dati collegati a un campo è importante quando un'applicazione esegue la convalida dei dati a livello di record in un gestore di evento di un dataset (come *BeforePost*). La convalida può essere eseguita sui campi di un record indipendentemente dal fatto che il fuoco si trovi sul relativo controllo associato ai dati. Nel caso la convalida fallisca per uno specifico campo del record, si può ottenere che il fuoco si sposti sul controllo associato ai dati che contiene dati non corretti, in modo che l'utente possa immettere le correzioni.

Il fuoco dei componenti associati ai dati di un campo si controlla col metodo *FocusControl* del campo. *FocusControl* imposta il fuoco sul primo controllo associato ai dati in una scheda associata a un campo. Prima di convalidare il campo, un gestore di evento dovrebbe chiamare il metodo *FocusControl* del campo. Il codice che segue

illustra come chiamare il metodo *FocusControl* per il campo *Company* nella tabella *Customer*::

```
CustomersCompany->FocusControl();
```

La tabella che segue elenca gli altri metodi del componente campo e il loro uso. Per un elenco completo e per informazioni particolareggiate sull'uso di ciascun metodo, consultare le voci relative a *TField* e ai suoi discendenti nella guida in linea *VCL Reference*.

Tabella 23.6 Altri metodi dei componenti campo

Metodo	Funzione
AssignValue	Imposta un valore di campo a un determinato valore usando una funzione di conversione automatica basata sul tipo del campo.
Clear	Svuota il campo e ne imposta il valore a NULL.
GetData	Recupera dal campo i dati non formattati.
IsValidChar	Determina se un carattere immesso da un utente in un controllo associato ai dati per impostare un valore è un carattere consentito.
SetData	Assegna al campo i dati non formattati.

Visualizzazione, conversione e accesso ai valori di campo

I controlli associati ai dati quali *TDBEdit* e *TDBGrid* visualizzano automaticamente i valori associati ai componenti campo. Se la modifica è consentita per il dataset e per i controlli, i controlli associati ai dati possono anche inviare valori nuovi e modificati al database. In generale, le proprietà e i metodi incorporati nei controlli associati ai dati consentono a questi ultimi di connettersi ai dataset, visualizzare i valori ed effettuare aggiornamenti senza richiedere ulteriori operazioni di programmazione. È consigliabile ricorrere ad essi ogni qualvolta sia possibile nelle applicazioni database. Per ulteriori informazioni sui controlli data-aware, consultare il Capitolo 19, "Uso dei controlli dati".

I controlli standard possono anche visualizzare e modificare valori di database associati ai componenti campo. Tuttavia, con i controlli standard potrebbe essere necessario scrivere del codice aggiuntivo. Ad esempio, quando si utilizzano controlli standard, è compito dell'applicazione rilevare quando è il momento di aggiornare i controlli perché i valori del campo sono stati modificato. Se il dataset ha un componente datasource, è possibile utilizzarne gli eventi per assolvere questo compito. In particolare, l'evento *OnDataChange* consente di sapere quando può essere necessario aggiornare il valore di un controllo e l'evento *OnStateChange* può essere d'aiuto nel determinare quando attivare o disattivare i controlli. Per ulteriori informazioni su questi eventi, consultare "Risposta alle modifiche mediate dal datasource" a pagina 19-4.

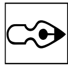
I seguenti paragrafi spiegano come operare con i valori dei campi in modo che sia possibile visualizzarli nei controlli standard.

Visualizzazione dei valori dei componenti campo nei controlli standard

Un'applicazione può accedere ai valori delle colonne di un datset mediante la proprietà *Value* di un componente campo. Ad esempio, il seguente gestore di evento *OnDataChange* aggiorna il testo in un controllo *TEdit* poiché il valore del campo *CustomersCompany* potrebbe essere stato cambiato:

```
void __fastcall TForm1::Table1DataChange(TObject *Sender, TField *Field)
{
  Edit3->Text = CustomersCompany->Value;
}
```

Questo metodo funziona bene per i valori stringa, ma può richiedere ulteriore programmazione per gestire le conversioni per altri tipi di dati. Fortunatamente, i componenti campo possiedono funzioni native per gestire le conversioni.



Per accedere ai valori dei campi e modificarli, è possibile anche ricorrere ai Variant. Per ulteriori informazioni sull'uso dei variant per accedere ai valori dei campi e impostarli, vedere "Accesso ai valori dei campi con la proprietà predefinita del dataset" a pagina 23-21.

Conversione dei valori di campo

Le proprietà di conversione cercano di convertire un tipo di dati in un altro. Per esempio, la proprietà *AsString* converte i valori numerici e booleani sotto forma di stringhe. La seguente tabella elenca le proprietà di conversione dei componenti campo e indica quali proprietà sono consigliate per ciascuna classe di componente campo::

	AsVariant	AsString	AsInteger	AsFloat AsCurrency AsBCD	AsDateTime AsSQLTimeStamp	AsBoolean
TStringField	Sì	ND	Sì	Sì	Sì	Sì
TWideStringField	Sì	Sì	Sì	Sì	Sì	Sì
TIntegerField	Sì	Sì	ND	Sì		
TSmallIntField	Sì	Sì	Sì	Sì		
TWordField	Sì	Sì	Sì	Sì		
TLargeintField	Sì	Sì	SìSì	Sì		
TFloatField	Sì	Sì	Sì	Sì		
TCurrencyField	Sì	Sì	Sì	Sì		
TBCDField	Sì	Sì	Sì	Sì		
TFMTBCDField	Sì	Sì	Sì	Sì		
TDateTimeField	Sì	Sì		Sì	Sì	
TDateField	Sì	Sì		Sì	Sì	
TTimeField	Sì	Sì		Sì	Sì	
TSQLTimeStampField	Sì	Sì		Sì	Sì	

TBooleanField	Sì	Sì				
TBytesField	Sì	Sì				
TVarBytesField	Sì	Sì				
TBlobField	Sì	Sì				
TMemoField	Sì	Sì				
TGraphicField	Sì	Sì				
TVariantField	ND	Sì	Sì	Sì	Sì	Sì
TAggregateField	Sì	Sì				

Si noti che alcune colonne nella tabella fanno riferimento a più proprietà di conversione (come *AsFloat*, *AsCurrency* e *AsBCD*). Questo è il motivo per cui tutti i tipi di dati di campo che supportano una di quelle proprietà supportano sempre anche le altre.

Si noti che la proprietà *AsVariant* può anche tradurre di tutti i tipi di dati. Per tutti gli altri tipi di dati non elencati in precedenza, è disponibile anche *AsVariant* (ed è, infatti, l'unica opzione). In caso di dubbio, utilizzare *AsVariant*.

In alcuni casi, le conversioni non sono possibili. Per esempio, *AsDateTime* può essere utilizzato per convertire una stringa in un formato date, time o datetime solo se il valore della stringa è in un formato datetime riconoscibile. Un tentativo di conversione non riuscito genera un'eccezione.

In alcuni altri casi, la conversione è possibile, ma i risultati non sono sempre intuitivi. Per esempio, che cosa significa convertire un valore *TDateTimeField* nel formato in virgola mobile? *AsFloat* converte la data riportata nel campo nel numero di giorni trascorsi dal 31/12/1899 e converte l'ora riportata nel campo in una frazione di 24 ore. La Tabella 23.7 elenca le conversioni consentite che producono risultati speciali:

Tabella 23.7 Risultati speciali di conversioni

Conversione	Risultato
da *String* a *Boolean*	Converte "True", "False", "Yes" e "No" in valore booleano. Gli altri valori sollevano eccezioni.
da *Float* a *Integer*	Arrotonda il valore in virgola mobile al valore intero più vicino.
da *DateTime* o da *SQLTimeStamp* a *Float*	Converte la data nel numero di giorni trascorsi dal 31/12/1899 e l'ora in una frazione di 24 ore.
da *Boolean* a *String*	Converte un valore booleano in "True" o "False."

Negli altri casi le conversioni non sono affatto possibili. In questi casi un eventuale tentativo di conversione solleva un'eccezione.

La conversione si verifica sempre prima che l'assegnazione venga eseguita. Per esempio, la seguente istruzione converte il valore di *CustomersCustNo* in una stringa e assegna quest'ultima al testo di un controllo di testo:

```
Edit1->Text = CustomersCustNo->AsString;
```

Viceversa, la seguente istruzione assegna il testo di un controllo di testo come numero intero al campo *CustomersCustNo*:

```
MyTableMyField->AsInteger = StrToInt(Edit1->Text);
```

Accesso ai valori dei campi con la proprietà predefinita del dataset

Il metodo più indicato per accedere al valore di un campo è l'uso di Variant con la proprietà *FieldValues*. Ad esempio, la seguente istruzione inserisce il valore di una casella di testo nel campo *CustNo* della tabella *Customers*:

```
Customers->FieldValues["CustNo"] = Edit2->Text;
```

Poiché la proprietà *FieldValues* è di tipo Variant, converte automaticamente gli altri tipi di dati in un valore Variant.

Per ulteriori informazioni sui Variant, vedere la Guida in linea.

Accesso ai valori dei campi con la proprietà Fields di un dataset

È possibile accedere al valore di un campo con la proprietà *Fields* del componente dataset al quale il campo appartiene.*Fields* gestisce una lista indicizzata di tutti i campi nel dataset. L'accesso ai valori dei campi con la proprietà *Fields* è utile quando occorre elaborare in successione numerose colonne o quando l'applicazione lavora con tabelle non disponibili allo sviluppatore in fase di progetto.

Per utilizzare la proprietà *Fields*, occorre conoscere l'ordine e i tipi di dati dei campi del dataset. Per specificare il campo cui accedere, si usa un numero ordinale. Il primo campo di un dataset corrisponde al numero 0. I valori dei campi devono essere convertiti nel modo appropriato utilizzando le proprietà di conversione di ciascun componente campo. Per ulteriori informazioni sulle proprietà di conversione dei componenti campo, vedere "Conversione dei valori di campo" a pagina 23-19.

Ad esempio, la seguente istruzione assegna il valore attivo della settima colonna (Country) della tabella *Customers* a un controllo di testo:

```
Edit1->Text = CustTable->Fields->Fields[6]->AsString;
```

Viceversa, un valore può essere assegnato a un campo impostando la proprietà *Fields* del dataset sul campo desiderato. Ad esempio:

```
Customers->Edit();
Customers->Insert();
Customers->Fields->Fields[6]->AsString = Edit1->Text;
Customers->Post();
```

Accesso ai valori dei campi con il metodo FieldByName di un dataset

Per accedere al valore di un campo, è possibile anche ricorrere al metodo *FieldByName* di un dataset. Tael metodo risulta utile quando è noto il nome del campo a cui accedere ma non si ha accesso alla tabella sottostante in fase di progetto.

Per utilizzare *FieldByName*, è necessario conoscere il dataset e il nome del campo a cui accedere. Il nome del campo viene passato come argomento del metodo. Per accedere al valore del campo o modificarlo, il risultato deve essere convertito con la proprietà appropriata per il componente campo, quale *AsString* o *AsInteger*. Per esempio, la seguente istruzione assegna il valore del campo *CustNo* del dataset *Customers* a un controllo di testo:

```
Edit2->Text = Customers->FieldByName(“CustNo”)->AsString;
```

Viceversa, la seguente istruzione assegna un valore a un campo:

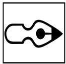
```
Customers->Edit();
Customers->FieldByName(“CustNo”)->AsString = Edit2->Text;
Customers->Post();
```

Impostazione del valore predefinito di un campo

Grazie alla proprietà *DefaultExpression*, è possibile specificare il modo in cui dovrebbe essere calcolato in esecuzione il valore predefinito di un campo di un dataset client o di un dataset BDE. *DefaultExpression* può essere qualsiasi espressione di valore SQL valida che non fa riferimento a valori di campo. Se l’espressione contiene letterali diversi da valori numerici, questi devono apparire tra virgolette. Ad esempio, il valore predefinito di mezzogiorno per un campo ora sarebbe

```
‘12:00:00’
```

compresi gli apici che racchiudono il valore costante.



Se la tabella del database sottostante stabilisce un valore predefinito per il campo, l’impostazione predefinita che si specifica in *DefaultExpression* ha la precedenza. Questo perché *DefaultExpression* viene applicato quando il dataset registra il record che contiene il campo, il che si verifica prima che il record modificato venga applicato al server di database.

Utilizzo dei vincoli

I componenti campo nei dataset client o nei dataset BDE possono utilizzare i vincoli dei server SQL. Inoltre, le applicazioni possono creare e usare vincoli custom per questi dataset che sono locali all’applicazione. Tutti i vincoli sono regole o condizioni che impongono un limite sulla visibilità o sull’intervallo dei valori che un campo può contenere.

Creazione di un vincolo custom

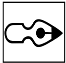
Un vincolo custom non viene importato dal server come gli altri vincoli. È un vincolo che viene dichiarato, implementato e applicato in una applicazione locale. Come tali, i vincoli custom possono essere utili per consentire una preconvalida dei dati immessi, ma un vincolo custom non può essere applicato a dati ricevuti da un’application server o ad esso inviati.

Per creare un vincolo custom, impostare la proprietà *CustomConstraint* specificando una condizione restrittiva e *ConstraintErrorMessage* specificando il messaggio da visualizzare quando un utente viola il vincolo in fase di esecuzione.

CustomConstraint è una stringa SQL che specifica tutte i particolari vincoli imposti da un'applicazione sul valore del campo. Per limitare i valori che l'utente può immettere in un campo, occorre impostare *CustomConstraint*. *CustomConstraint* può essere qualsiasi espressione di ricerca SQL valida, come:

```
x > 0 and x < 100
```

Il nome usato per fare riferimento al valore del campo può essere qualsiasi stringa che non sia una parola SQL riservata, purché lo si usi coerentemente in tutta l'espressione restrittiva.



I vincoli custom sono solo disponibili solo nei dataset BDE e nei dataset client.

I vincoli custom vengono imposti, in aggiunta a tutti gli altri vincoli, al valore del campo proveniente dal server. Per vedere i vincoli imposti dal server, consultare la proprietà *ImportedConstraint*.

Uso dei vincoli del server

La maggior parte dei database SQL di produzione usano vincoli per imporre condizioni sui possibili valori di un campo. Per esempio, un campo può non consentire valori NULL, può richiedere che il suo valore sia unico per quella colonna o che i suoi valori siano maggiori di *0* e minori di *150*. Benché sia possibile replicare tali condizioni nelle applicazioni client, i dataset client e i dataset BDE mettono a disposizione la proprietà ImportedConstraintper propagare localmente i vincoli di un server.

ImportedConstraint è una proprietà a sola lettura che specifica una clausola SQL che limita i valori di campo in un determinato modo. Ad esempio:

```
Value > 0 and Value < 100
```

Non si deve modificare il valore *ImportedConstraint*, tranne nel caso in cui si debbano modificare istruzioni SQL non standard o specifiche del server, importate come commento perché non interpretate dal motore di database.

Per aggiungere altri vincoli sul valore del campo, usare la proprietà *CustomConstraint*. I vincoli custom vengono imposti in aggiunta ai vincoli importati. Se si modificano i vincoli del server, si modifica anche il valore di *ImportedConstraint* ma i vincoli immessi nella proprietà *CustomConstraint* persistono.

La rimozione dei vincoli dalla proprietà *ImportedConstraint* non modificherà la validità dei valori di campo che violano quei vincoli. La rimozione dei vincoli implica che essi saranno controllati dal server anziché a livello locale. Quando i vincoli sono controllati localmente e si verifica una violazione della restrizione, invece di visualizzare un messaggio di errore proveniente dal server, verrà visualizzato il messaggio di errore impostato tramite la proprietà *ConstraintErrorMessage*.

Uso dei campi oggetto

I campi oggetto sono campi che rappresentano un insieme di altri tipi di dati più semplici. Questi includono campi ADT (Abstract Data Type) campi Array, campi DataSet, e campi Reference. Questi tipi di campo contengono o fanno riferimento ai campi figlio e ad altri dataset.

Un campo ADT e un campo array, contengono campi figlio. I campi figlio di un campo ADT possono essere un qualsiasi tipo scalare od oggetto (vale a dire, qualsiasi altro tipo di campo). Il tipo di ognuno di questi campi figlio può essere diverso da quello dell'altro. Un campo array contiene un array di campi figlio, tutti dello stesso tipo.

I campi dataset e reference sono campi che accedono ad altri dataset. Un campo riferimento memorizza un puntatore (riferimento) a un altro oggetto persistente (ADT).

Tabella 23.8 Tipi di componenti campo oggetto

Nome del componente	Funzione
TADTField	Rappresenta un campo ADT (Abstract Data Type).
TArrayField	Rappresenta un campo array.
TDataSetField	Rappresenta un campo che contiene un riferimento dataset annidato.
TReferenceField	Rappresenta un campo REF, un puntatore a un ADT.

Quando si aggiungono campi con il Fields editor a un dataset contenente campi oggetto, vengono creati automaticamente i campi oggetto persistenti di tipo corretto. L'aggiunta di campi oggetto persistenti a un dataset imposta automaticamente la proprietà *ObjectView* del dataset a **true**, il che permette al dataset di memorizzare questi campi in ordine gerarchico, piuttosto che esplicitarli come se i campi figlio costituenti fossero campi separati e indipendenti.

Le seguenti proprietà sono comuni a tutti i campi oggetto e forniscono le funzionalità per la gestione dei campi figlio e dei dataset.

Tabella 23.9 Proprietà comuni dei discendenti dei campi oggetto

Proprietà	Funzione
Fields	Contiene i campi figlio che appartengono al campo oggetto.
ObjectType	Classifica il campo oggetto.
FieldCount	Numero di campi figlio che appartengono al campo oggetto.
FieldValues	Fornisce l'accesso ai valori dei campi figlio.

Visualizzazione di campi ADT e array

Sia i campi ADT sia i campi array contengono campi figlio che possono essere visualizzati tramite controlli associati ai dati.

I controlli data-aware come *TDBEdit* che rappresentano un singolo valore di campo mostrano i valori dei campi figli campo una stringa delimitata con virgole non modificabile. Inoltre, se si imposta la proprietà *DataField* del controllo al campo figlio invece che allo stesso campo oggetto stesso, il campo figlio può essere visualizzato e modificato esattamente come qualsiasi altro normale campo dati.

Un controllo *TDBGrid* visualizza in modo diverso i campi ADT e i campi array, a seconda del valore della proprietà *ObjectView* del dataset. Quando *ObjectView* è **false**, ogni campo figlio appare in una colonna singola. Quando *ObjectView* è **true**, un campo ADT o un campo array possono essere espansi e ridotti facendo clic sulla freccia nella barra del titolo della colonna. Quando il campo viene espanso, ogni campo figlio appare nella propria colonna e barra del titolo, esattamente sotto la barra del titolo dello stesso ADT o array. Quando l'ADT o l'array viene ridotto, appare una sola colonna con una stringa non modificabile, delimitata da virgole, che contiene i campi figlio.

Uso dei campi ADT

Gli ADT sono tipi definiti dall'utente creati sul server e sono simili alle strutture. Un ADT può contenere diversi tipi di campo scalari, campi array, campi riferimento e ADT annidati.

Ci sono diversi modi per accedere ai dati nei tipi di campo ADT. Gli esempi seguenti assegnano un valore di campo figlio a una casella di modifica di nome *CityEdit*, e utilizzano la seguente struttura ADT:

```
Address
  Street
  City
  State
  Zip
```

Utilizzo di componenti campo persistenti

Il modo più semplice per accedere a valori di un campo ADT consiste nell'utilizzare componenti campo persistenti. Per la struttura ADT vista in precedenza, è possibile aggiungere alla tabella *Customer* i seguenti campi persistenti utilizzando il Fields editor:

```
CustomerAddress: TADTField;
CustomerAddrStreet: TStringField;
CustomerAddrCity: TStringField;
CustomerAddrState: TStringField;
CustomerAddrZip: TStringField;
```

Dati questi campi persistenti, è possibile accedere semplicemente ai campi figlio di un campo ADT mediante il nome:

```
CityEdit->Text = CustomerAddrCity->AsString;
```

Benché i campi persistenti siano il modo più semplice per accedere a campi figlio di ADT, non è possibile utilizzarli se, in fase di progettazione, non si conosce la struttura

del dataset. Quando si accede ai campi figli di ADT senza utilizzare campi persistenti, è necessario impostare a **True** la proprietà *ObjectView* del dataset.

Utilizzo del metodo FieldByName del dataset

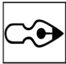
È possibile accedere ai figli di un campo ADT utilizzando il metodo *FieldByName* del dataset, qualificando il nome del campo figlio con il nome del campo ADT:

```
CityEdit->Text = Customer->FieldByName(“Address.City”)->AsString;
```

Utilizzo della proprietà FieldValues del dataset

È possibile utilizzare nomi di campo qualificati anche con la proprietà *FieldValues* di un dataset:

```
CityEdit->Text = Customer->FieldValues["Address.City"];
```



A differenza di altri metodi di esecuzione per accedere ai valori dei campo figlio di ADT, la proprietà *FieldValues* funziona anche se la proprietà *ObjectView* del dataset è impostata a **false**.

Utilizzo della proprietà FieldValues del campo ADT

Tramite la proprietà *FieldValues* di *TADTField* è possibile accedere al valore di un campo figlio. *FieldValues* accetta e restituisce un *Variant*, cosicché può gestire e convertire campi di qualunque tipo. Il parametro dell'indice è un valore integer che specifica lo scostamento del campo.

```
CityEdit->Text = ((TADTField*)Customer->FieldByName("Address"))->FieldValues[1];
```

Utilizzo della proprietà Fields del campo ADT

Ogni campo ADT ha una proprietà *Fields* analoga alla proprietà *Fields* di un dataset. Come la proprietà *Fields* di un dataset, è possibile utilizzarla per accedere a campi figlio in base alla loro posizione:

```
CityEdit->Text = ((TADTField*)Customer->FieldByName("Address"))->Fields->
Fields[1]->AsString;
```

oppure in base al nome:

```
CityEdit->Text = ((TADTField*)Customer->FieldByName("Address"))->
Fields->FieldByName("City")->AsString;
```

Uso dei campi array

I campi array sono costituiti da un gruppo di campi dello stesso tipo. Il tipo di campo può essere scalare (per esempio, float o string), o non scalare (un ADT), ma non può essere un campo array di array. La proprietà SparseArrays di *TDataSet* determina se per ciascun elemento del campo array viene creato un oggetto *TField* univoco.

Esistono diversi modi per accedere ai dati nei tipi di campo array. Se non si utilizzano campi persistenti, prima di potere accedere agli elementi di un campo array è necessario impostare a **True** la proprietà *ObjectView* del dataset.

Utilizzo di campi persistenti

È possibile mappare i campi persistenti su singoli elementi dell'array di un campo array. Ad esempio, si prenda in esame un campo array *TelNos_Array*, che è un array di stringhe con sei elementi. I campi persistenti seguenti, creati per il componente tabella *Customer*, rappresentano il campo *TelNos_Array* e i suoi sei elementi:

```
CustomerTELNOS_ARRAY: TArrayField;
CustomerTELNOS_ARRAY0: TStringField;
CustomerTELNOS_ARRAY1: TStringField;
CustomerTELNOS_ARRAY2: TStringField;
CustomerTELNOS_ARRAY3: TStringField;
CustomerTELNOS_ARRAY4: TStringField;
CustomerTELNOS_ARRAY5: TStringField;
```

Dati questi campi persistenti, il seguente codice utilizza un campo persistente per assegnare il valore di un elemento dell'array a una casella di testo di nome *TelEdit*.

```
TelEdit->Text = CustomerTELNOS_ARRAY0->AsString;
```

Uso della proprietà FieldValues del campo array

Tramite la proprietà *FieldValues* del campo array è possibile accedere al valore di un campo figlio. *FieldValues* accetta e restituisce un *Variant*, pertanto può gestire e convertire campi di qualunque tipo. Ad esempio:

```
TelEdit->Text = ((TArrayField*)Customer->FieldByName("TelNos_Array"))->FieldValues[1];
```

Uso della proprietà Fields del campo array

TArrayField ha una proprietà *Fields* che è possibile utilizzare per accedere ai singoli sottocampi. Il procedimento è illustrato nel codice seguente, in cui un campo array (*OrderDates*) viene utilizzato per riempire una casella di riepilogo con elementi di array non nulli:

```
for (int i = 0; i < OrderDates->Size; ++i)
  if (!OrderDates->Fields->Fields[i]->IsNull)
    OrderDateListBox->Items->Add(OrderDates->Fields->Fields[i]->AsString);
```

Uso dei campi dataset

I campi dataset forniscono l'accesso ai dati memorizzati in un dataset annidato. La proprietà NestedDataSet referenzia il dataset annidato. Ai dati del dataset annidato si può poi accedere tramite gli oggetti campo del dataset stesso.

Visualizzazione dei campi dataset

I controlli *TDBGrid* consentono la visualizzazione dei dati memorizzati nei campi del dataset. In un controllo *TDBGrid*, un campo del dataset viene indicato in ciascuna cella di una colonna del dataset mediante la stringa "(DataSet)" e, in fase di esecuzione, alla destra del campo appare anche un pulsante ellissi. Facendo clic sul pulsante si richiama una nuova scheda con una griglia che visualizza il dataset associato al campo dataset del record attivo. Questa scheda può anche essere richiamata da programma con il metodo *ShowPopupEditor* della griglia DB. Per

esempio, se la settima colonna della griglia rappresenta un campo dataset, il codice seguente visualizzerà il dataset associato a quel campo per il record attivo.

```
DBGrid1->ShowPopupEditor(DBGrid1->Columns->Items[7], -1, -1);
```

Accesso ai dati in un dataset annidato

Di solito un campo di un dataset non è collegato direttamente a un controllo associato ai dati. Piuttosto, poiché un dataset annidato non è altro che un dataset, il mezzo per accedere ai dati in esso contenuti è tramite un discendente di *TDataSet*. Il tipo di dataset che si utilizza è determinato dal dataset genitore (quello con il campo dataset). Ad esempio, un dataset BDE utilizza *TNestedTable* per rappresentare i dati nei campi del proprio dataset, mentre i dataset client utilizzano altri dataset client.

Per accedere ai dati in un campo dataset:

1 Creare un oggetto persistente *TDataSetField* richiamando il Fields editor per il dataset genitore.

2 Creare un dataset per rappresentare i valori in quel campo dataset. Deve essere di un tipo compatibile con il dataset genitore.

3 Impostare la proprietà *DataSetField* del dataset creato al punto 2 al campo dataset persistente creato al punto 1.

Se il campo dataset annidato del record corrente ha un valore, il componente dataset di dettaglio conterrà record con dati annidati; in caso contrario, il dataset di dettaglio sarà vuoto.

Before inserting records into a nested dataset, you should be sure to post the corresponding record in the master table, if it has just been inserted. If the inserted record is not posted, it will be automatically posted before the nested dataset.

Uso dei campi riferimento

I campi riferimento memorizzano un puntatore o un riferimento a un altro oggetto ADT. Questo oggetto ADT è un singolo record di un'altra tabella oggetto. I campi riferimento si riferiscono sempre a un unico record di un dataset (tabella oggetto). I dati dell'oggetto referenziato vengono di fatto restituiti in un dataset annidato, ma è possibile accedervi tramite la proprietà *Fields* in *TReferenceField*.

Visualizzazione dei campi riferimento

In un controllo *TDBGrid*, un campo riferimento viene rappresentato, in ciascuna cella della colonna del dataset, con la scritta (Reference) e, in fase di esecuzione, tramite un pulsante con puntini di sospensione a destra. In fase di esecuzione, facendo clic sul pulsante si richiama una nuova scheda con una griglia che visualizza l'oggetto associato al campo riferimento del record attivo.

Questa scheda può anche essere richiamata da programma con il metodo *ShowPopupEditor* della griglia DB. Per esempio, se la settima colonna della griglia rappresenta un campo riferimento, il codice seguente visualizzerà l'oggetto associato a quel campo per il record attivo.

```
DBGrid1->ShowPopupEditor(DBGrid1->Columns->Items[7], -1, -1);
```

Accesso ai dati di un campo riferimento

È possibile accedere ai dati in un campo riferimento in modo analogo a quello utilizzato per accedere a un dataset annidato:

1. Creare un oggetto persistente *TDataSetField* richiamando il Fields editor per il dataset genitore.
2. Creare un dataset per rappresentare il valore di quel campo di dataset.
3. Impostare la proprietà *DataSetField* del dataset creato al punto 2 al campo dataset persistente creato al punto 1.

Se il riferimento è assegnato, il dataset conterrà un unico record con i dati referenziati. Se è nullo, il dataset di riferimento sarà vuoto.

È possibile anche utilizzare la proprietà Fields del campo riferimento per accedere ai dati in un campo riferimento. L'esempio seguente assegna i dati del campo riferimento *CustomerRefCity* a una casella di modifica di nome *CityEdit*:

```
CityEdit->Text = CustomerADDRESS_REF->NestedDataSet->Fields->Fields[1]->AsString;
```

Quando si modificano i dati in un campo riferimento, di fatto vengono modificati i dati referenziati.

Per assegnare un campo riferimento, bisogna prima usare un'istruzione SELECT per selezionare il riferimento dalla tabella. Ad esempio:

```
AddressQuery->SQL->Text = "SELECT REF(A) FROM AddressTable A WHERE A.City = 'San
Francisco'";
AddressQuery->Open();
CustomerAddressRef->Assign(AddressQuery->Fields->Fields[0]);
```

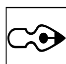
Capitolo

24

Uso di Borland Database Engine

Borland Database Engine (BDE) è un meccanismo di accesso ai dati condivisibile da più applicazioni. Definisce una potente libreria di chiamate API che servono a creare, a ristrutturare, a prelevare dati da server di database locali e remoti, ad aggiornarli o comunque a manipolarli. BDE offre un'interfaccia uniforme per l'accesso a un'ampia varietà di server di database, utilizzando dei driver per collegarsi a vari database. A seconda della versione di C++Builder disponibile, si utilizzeranno i driver per database locali (Paradox, dBASE, FoxPro e Access), i driver SQL Links per server di database remoti come InterBase, Oracle, Sybase, Informix, server Microsoft SQL e DB2, più un adattatore ODBC che consente di fornire driver ODBC realizzati in proprio.

Per la distribuzione di applicazioni BDE, è necessario distribuire BDE con l'applicazione. La dimensione e la complessità dell'applicazione aumentano, ma sono possibili sia l'uso condiviso con altre applicazioni BDE sia un'ampia gamma di funzioni ausiliarie per la manipolazione dei database. Benché sia possibile utilizzare l'API di BDE direttamente nelle applicazioni, i componenti alla pagina BDE della Component palette si incaricano di svolgere la maggior parte di queste funzionalità .



Per informazioni sull'API BDE, consultare il file di guida in linea, BDE32.hlp, che si trova nella directory dove è stato installato Borland Database Engine.

L'architettura BDE

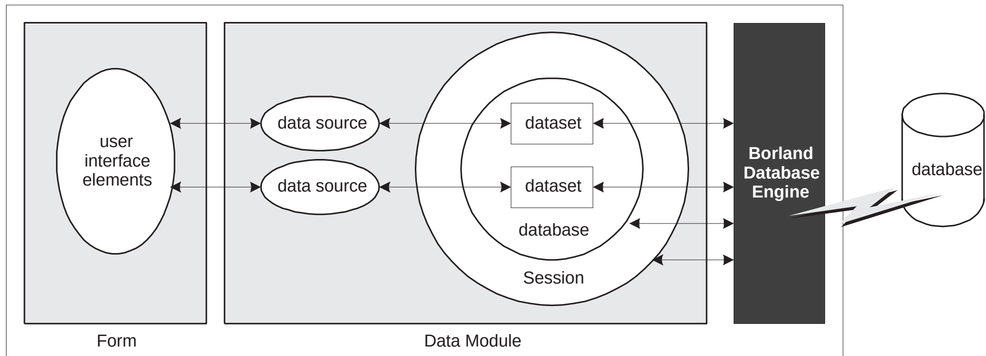
Le applicazioni che utilizzano BDE sfruttano una variante dell'architettura generale di database descritta in "Architettura di un database" a pagina 18-6. Oltre agli elementi dell'interfaccia utente, al datasource e ai dataset comuni a tutte le applicazioni database C++Builder , le applicazioni BDE possono comprendere

- Uno o più componenti database per il controllo delle transazioni e la gestione delle connessioni con il database.

- Uno o più componenti session per isolare le operazioni di accesso ai dati come le connessioni con il database e la gestione di gruppi di database.

Le relazioni tra i componenti di un'applicazione BDE sono illustrate in Figura 24.1:

Figura 24.1 Componenti in un'applicazione BDE



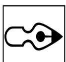
Impiego di dataset per BDE

I dataset utilizzano Borland Database Engine (BDE) per accedere ai dati. Essi ereditano le funzionalità dei dataset comuni descritte in Capitolo 22, "I dataset", mentre sfruttano BDE per l'implementazione. Inoltre, tutti i dataset per BDE aggiungono proprietà , eventi e metodi per le seguenti funzioni:

- Associazione di un dataset a connessioni di database e di sessione.
- Cache dei BLOB.
- Ottenimento degli handle BDE.

Ci sono tre dataset per BDE:

- *TTable*, un dataset di tipo tabella che rappresenta tutte le righe e le colonne di una singola tabella di database. Per la descrizione delle funzioni comuni ai dataset tabellari fare riferimento a "Utilizzo dei dataset di tipo tabella" a pagina 22-26. Per la descrizione delle funzioni univoche a *TTable*, fare riferimento a "Utilizzo di TTable" a pagina 24-5.
- *TQuery*, un dataset di tipo query che incorpora un'istruzione SQL e permette alle applicazioni di accedere ai record risultanti, se esistono. Consultare "Utilizzo di dataset di tipo query" a pagina 22-43 per la descrizione delle funzioni comuni ai dataset di tipo query. Consultare "Utilizzo di TQuery" a pagina 24-9 per la descrizione delle funzioni speciali di *TQuery*.
- *TStoredProc*, un dataset di tipo procedura registrata che esegue una procedura registrata definita su un server di database. Consultare "Utilizzo di dataset di tipo procedura registrata" a pagina 22-52 per una descrizione delle funzioni comuni ai dataset di tipo procedura registrata. Per la descrizione delle funzioni uniche di *TStoredProc* consultare "Utilizzo di TStoredProc" a pagina 24-12.



Oltre ai tre tipi di dataset per BDE, esiste un dataset client basato su BDE (*TBDEClientDataSet*) utilizzabile per la gestione degli aggiornamenti in cache. Per le

informazioni relative, consultare "Utilizzo di un dataset client per registrare in cache gli aggiornamenti" a pagina 27-17.

Associazione di un dataset a connessioni di database e di sessione

Per prelevare i dati da un server di database, i dataset BDE utilizzano sia un database che una sessione.

- I database rappresentano connessioni a specifici server di database. Il database identifica un certo driver BDE, un particolare server di database che lo utilizza, e un set di parametri per la connessione al server di database. Ogni database è rappresentato da un componente *TDatabase*. Si possono associare i dataset a un componente *TDatabase* che viene aggiunto a una scheda o a un modulo dati, oppure semplicemente, si identifica il server di database tramite il nome e si lascia che sia C++Builder a generare un componente database implicito. L'impiego di componenti *TDatabase* creati esplicitamente è raccomandato nella maggior parte delle applicazioni, perché essi consentono di controllare meglio le modalità di connessione, compreso il processo di login, e permettono al programmatore di creare e di utilizzare le transazioni.

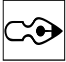
  Per associare un dataset per BDE a un database, si ricorre alla proprietà *DatabaseName*. DatabaseName è una stringa contenente informazioni diverse, a seconda che si utilizzi un componente database esplicito, oppure, in caso contrario, il tipo di database che si utilizza:

  - Se si utilizza un componente esplicito *TDatabase*, *DatabaseName* è il valore della sua proprietà *DatabaseName*.
  - Se si vuole usare un componente database implicito, e il database ha un alias BDE, si può specificare, l'alias BDE come valore di *DatabaseName*,. Un alias BDE rappresenta un database, più informazioni relative alla sua configurazione. Le informazioni di configurazione associate a un alias sono diverse a seconda del tipo di database (Oracle, Sybase, InterBase, Paradox, dBASE e così via). Per creare e gestire gli alias BDE si ricorre allo strumento BDE Administration o a SQL Explorer.
  - Volendo utilizzare un componente database implicito con un database Paradox o dBASE, è possibile anche ricorrere a *DatabaseName* e specificare semplicemente la directory dove sono situate le tabelle di database.

- In un'applicazione, una sessione fornisce la gestione globale di un gruppo di connessioni di database. Quando all'applicazione si aggiungono dei dataset BDE, questa conterrà automaticamente un componente session, chiamato *Session*. Il database e i componenti dataset aggiunti all'applicazione sono associati automaticamente a questa sessione predefinita. Viene anche controllato l'accesso ai file Paradox protetti da password e vengono specificate le directory per la condivisione di file Paradox in rete. Le connessioni al database e l'accesso ai file Paradox si controllano utilizzando le proprietà , gli eventi e i metodi della sessione.

  Si può utilizzare la sessione predefinita per controllare tutte le connessioni al database nell'applicazione. In alternativa, si aggiungono ulteriori componenti in progettazione o si creano dinamicamente in esecuzione per controllare un

sottoinsieme di connessioni al database. Per associare il dataset a un componente session esplicitamente creato, si ricorre alla proprietà *SessionName*. Se nell'applicazione non si utilizzano componenti session espliciti, non è necessario fornire il valore di questa proprietà . Se si utilizza la sessione predefinita, o si specifica esplicitamente una sessione mediante la proprietà *SessionName*, è possibile accedere alla sessione associata a un dataset leggendo la proprietà *DBSession*.



Se si utilizza un componente session, la proprietà *SessionName* di un dataset deve corrispondere alla proprietà *SessionName* del componente database a cui il dataset è associato.

Per ulteriori informazioni su *TDatabase* e su *TSession*, consultare "Collegamento ai database tramite TDatabase" a pagina 24-13 e "Gestione delle sessioni di database" a pagina 24-17.

Cache dei BLOB

I dataset BDE hanno tutti una proprietà *CacheBlobs* che controlla se BDE registra a livello locale i campi, quando un'applicazione legge i record BLOB. Per impostazione predefinita, *CacheBlobs* è **true**, il che significa che BDE memorizza nella cache una copia locale dei campi BLOB. La registrazione nella cache dei campi BLOB migliora le prestazioni dell'applicazione in quanto BDE memorizza localmente le copie dei BLOB, invece di prelevarli ripetutamente dal server di database mano a mano che l'utente fa scorrere i record.

Nelle applicazioni e negli ambienti dove i BLOB si aggiornano o si sostituiscono spesso e dove una vista fresca dei dati BLOB è più importante delle prestazioni, è possibile impostare *CacheBlobs* a **false** per assicurarsi che venga sempre vista la versione più recente di un campo BLOB.

Ottenimento degli handle BDE

È possibile utilizzare dei dataset per BDE senza fare chiamate dirette alla API di Borland Database Engine. I dataset per BDE, combinati con componenti database e di sessione, incorporano molte funzionalità BDE. Tuttavia, se è necessario fare chiamate dirette alla API di BDE, per le risorse gestite da BDE servono gli handle BDE. Molte API BDE richiedono questi handle come parametri.

Tutti i dataset per BDE includono tre proprietà a sola lettura per l'accesso in fase di esecuzione agli handle BDE:

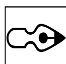
- *Handle* è un handle al cursor BDE che accede ai record nel dataset.
- *DBHandle* è un handle al database che contiene le tabelle o la procedura registrata relative.
- *DBLocale* è un handle al driver BDE del linguaggio del dataset e controlla la sequenza di ordinamento e il set di caratteri utilizzati per i dati di stringa.

Queste proprietà sono assegnate automaticamente a un dataset quando questo si collega a un server di database tramite BDE. Per ulteriori informazioni sulla API di BDE, consultare il file della guida in linea, BDE32.HLP.

Utilizzo di TTable

TTable incapsula la struttura completa e i dati in una tabella di database e implementa tutte le funzionalità di base introdotte da *TDataSet*, come pure tutte le funzioni speciali tipiche dei dataset di tipo tabella. Prima di analizzare le funzionalità esclusive di *TTable*, conviene familiarizzare con le funzionalità comuni dei database descritte in "I dataset", compresa la sezione sui dataset tabellari che inizia a pagina 22-26.

TTable è un dataset per BDE, e quindi deve essere associato a un database e a una sessione. Il paragrafo "Associazione di un dataset a connessioni di database e di sessione" a pagina 24-3 spiega come si formano queste associazioni. Una volta che il dataset è associato a un database e a una sessione, è possibile legarlo a una particolare tabella di database impostando la proprietà *TableName* e, se si utilizza una tabella di testo ASCII Paradox, dBASE, FoxPro o comma-delimited, alla proprietà *TableType*.



La tabella deve essere chiusa quando si modifica la sua associazione a un database, a una sessione o a una tabella di database, o quando si imposta la proprietà *TableType*. Tuttavia, prima di chiudere la tabella per modificare queste proprietà , bisogna registrare o annullare le modifiche in attesa. Se sono abilitati gli aggiornamenti in cache, bisogna chiamare il metodo *ApplyUpdates* per scrivere nel database le modifiche richieste.

I componenti *TTable* offrono un supporto ineguagliabile per la gestione delle tabelle di database locali (Paradox, dBASE, FoxPro e tabelle di testo ASCII comma-delimited). Qui di seguito sono riportati le proprietà e i metodi speciali che implementano questo supporto.

Inoltre, i componenti *TTable* possono sfruttare il supporto BDE per le operazioni in batch (operazioni a livello di tabella per aggiungere, aggiornare, cancellare o copiare interi gruppi di record). L'argomento è trattato in "Importazione di dati da altre tabelle" a pagina 24-8.

Specifica del tipo di tabella per tabelle locali

Se un'applicazione accede a tabelle Paradox, dBASE, FoxPro o di testo ASCII comma-delimited, allora BDE utilizza la proprietà *TableType* per determinare il tipo di tabella (la sua struttura prevista). *TableType* non viene usato quando *TTable* rappresenta una tabella a base SQL su un server di database.

Per impostazione predefinita, *TableType* è impostato a *ttDefault*. In questo caso, BDE determina il tipo di tabella dalla sua estensione. La Tabella 24.1 riporta le estensioni riconosciute da BDE, e le ipotesi che fa sul tipo della tabella:

Tabella 24.1 Tipi di tabella riconosciuti da BDE in base all'estensione del file

Estensione	Tipo di tabella
No file extension	Paradox
.DB	Paradox

Tabella 24.1 Tipi di tabella riconosciuti da BDE in base all'estensione del file (continua)

Estensione	Tipo di tabella
.DBF	dBASE
.TXT	ASCII text

Se le tabelle di testo Paradox, dBASE e ASCII locali utilizzano le estensioni file come descritto in Tabella 24.1, è possibile lasciare *TableType* impostato su *ttDefault*. Altrimenti, l'applicazione deve impostare *TableType* in modo da indicare il tipo di tabella corretto. In Tabella 24.2 sono indicati i valori che è possibile assegnare a *TableType*:

Tabella 24.2 Valori di TableType

Valore	Tipo di tabella
ttDefault	Tipo di tabella determinato automaticamente da BDE
ttParadox	Paradox
ttDBase	dBASE
ttFoxPro	FoxPro
ttASCII	Testo ASCII Comma-delimited

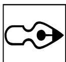
Controllo dell'accesso a tabelle locali in lettura/scrittura

Come qualsiasi dataset di tipo tabella, *TTable* permette di controllare l'accesso in lettura e in scrittura da parte dell'applicazione tramite la proprietà *ReadOnly*.

Inoltre, nel caso di tabelle Paradox, dBASE e FoxPro, *TTable* consente di controllare l'accesso in lettura e in scrittura alle tabelle da parte di altre applicazioni. La proprietà *Exclusive* determina se l'applicazione ottiene l'accesso esclusivo in lettura/scrittura a una tabella Paradox, dBASE o FoxPro. Allo scopo, si imposta a **true** la proprietà *Exclusive* del componente tabella prima di aprire la tabella. Se si riesce ad aprire una tabella per accesso esclusivo, le altre applicazioni non possono né leggere né scrivere dati nella tabella. La richiesta di accesso esclusivo non viene concessa quando la tabella è già in uso.

Le seguenti istruzioni aprono una tabella per accesso esclusivo:

```
CustomersTable->Exclusive = true; // Set request for exclusive lock
CustomersTable->Active = true; // Now open the table
```



È possibile tentare di impostare la proprietà *Exclusive* su tabelle SQL, ma alcuni server non supportano il blocco esclusivo a livello di tabella. Altri server concedono il blocco esclusivo, ma consentono ad altre applicazioni di leggere i dati dalla tabella. Per ulteriori informazioni sul blocco esclusivo di tabelle di database su un server, consultare la documentazione del server.

Specifica di file indice dBASE

Nella maggior parte dei server, per specificare un indice si utilizzano i metodi comuni a tutti i dataset tabellari, descritti in "Ordinamento dei record con indici" a pagina 22-27.

Tuttavia, per le tabelle dBASE che utilizzano file indice aggiunti, oppure indici tipo dBASE III PLUS (*.NDX), è necessario invece utilizzare le proprietà *IndexFiles* e *IndexName.* Alla proprietà *IndexFiles* si assegna il nome del file indice aggiunto, o si elencano i file.NDX. Poi si specifica un indice nella proprietà *IndexName* per fare in modo che ordini attivamente il dataset.

In fase di progetto, si fa clic sul pulsante ellissi (...)nel valore della proprietà *IndexFiles* di Object Inspector per richiamare l'Index Files editor. Per aggiungere un file indice aggiunto o un file .NDX si fa clic sul pulsante Add nella finestra di dialogo Index Files e si seleziona il file nella finestra di dialogo Open. La procedura va ripetuta per ogni file indice non di produzione o .NDX. Dopo avere aggiunto tutti gli indici desiderati si fa clic sul pulsante OK della finestra di dialogo Index Files.

La stessa operazione può essere eseguita da programma in fase di esecuzione. Allo scopo, si accede alla proprietà *IndexFiles* utilizzando le proprietà e i metodi delle liste di stringhe. Quando si aggiunge un nuovo gruppo di indici, prima si chiama il metodo *Clear* della proprietà *IndexFiles* della tabella per rimuovere eventuali voci inserite. Si chiama il metodo *Add* per aggiungere ogni file indice non di produzione o file .NDX:

```
Table2->IndexFiles->Clear();
Table2->IndexFiles->Add("Bystate.ndx");
Table2->IndexFiles->Add("Byzip.ndx");
Table2->IndexFiles->Add("Fullname.ndx");
Table2->IndexFiles->Add("St_name.ndx");
```

Dopo avere aggiunto gli eventuali file indice non di produzione o .NDX, diventano disponibili i nomi dei singoli indici nel file indice ed è possibile assegnarli alla proprietà *IndexName*. I tag indice vengono elencati anche quando si utilizza il metodo *GetIndexNames* ed quando si esaminano le definizioni di indice attraverso gli oggetti *TIndexDef* nella proprietà *IndexDefs*. I file .NDX elencati correttamente vengono aggiornati automaticamente mano a mano che i dati vengono aggiunti, modificati o cancellati nella tabella (indipendentemente dal fatto che la proprietà *IndexName* utilizzi o meno un certo indice).

Nell'esempio qui riportato, il componente *IndexFiles* del componente tabella *AnimalsTable* è impostato al file indice non di produzione ANIMALS.MDX e quindi la sua proprietà *IndexName* è impostata al tag indice chiamato "NAME":

```
AnimalsTable->IndexFiles->Add("ANIMALS.MDX");
AnimalsTable->IndexName = "NAME";
```

Una volta che si è specificato il file indice, l'utilizzo di indici non di produzione o .NDX non differisce da quello di qualsiasi altro indice. Specificando un nome indice si ordinano i dati nella tabella rendendoli disponibili per ricerche indicizzate, per intervalli e (nel caso di indici non di produzione) per collegamenti master-detail. Per i dettagli su questo modo di usare gli indici, consultare "Utilizzo dei dataset di tipo tabella" a pagina 22-26.

Quando si utilizzano indici .NDX, in stile dBASE III, con componenti *TTable* vanno tenute presenti due considerazioni particolari. La prima è che i file .NDX non si possono usare come base per collegamenti master-detail. La seconda è che quando si attiva un indice .NDX con la proprietà *IndexName*, è necessario includere l'estensione .NDX nel valore della proprietà come parte del nome dell'indice:

```
Table1->IndexName = "ByState.NDX";
TVarRec vr = ("NE");
Table1->FindKey(&vr, 0);
```

Come rinominare tabelle locali

Per rinominare in fase di progettazione le tabelle Paradox o dBASE, si fa clic con il pulsante destro sul componente tabella e si seleziona Rename Table nel menu di scelta rapida.

Per rinominare in fase di esecuzione le tabelle Paradox o dBASE, si chiama il metodo *RenameTable* della tabella. Ad esempio, con la seguente istruzione si rinomina la tabella Customer in CustInfo:

```
Customer->RenameTable("CustInfo");
```

Importazione di dati da altre tabelle

Per importare dati da un'altra tabella si utilizza il metodo *BatchMove* di un componente tabella. *BatchMove* può

- Copiare nella tabella attiva dei record da un'altra tabella.
- Aggiornare nella tabella attiva dei record che si trovano in un'altra tabella.
- Aggiungere alla fine della tabella attiva dei record di un'altra tabella.
- Cancellare nella tabella attiva dei record che si trovano in un'altra tabella.

BatchMove accetta due parametri: il nome della tabella da cui importare i dati e una specifica di modo che determina quale operazione di importazione eseguire. La Tabella 24.3 mostra le possibili impostazioni della specifica di modo:

Tabella 24.3 Modalità di importazione di BatchMove

Valore	Significato
batAppend	Aggiunge tutti i record dalla tabella sorgente alla fine della tabella attiva.
batAppendUpdate	Aggiunge tutti i record dalla tabella sorgente alla fine della tabella attiva e aggiorna nella tabella attiva i record esistenti con record corrispondenti della tabella sorgente.
batCopy	Copia nella tabella attiva tutti i record della tabella sorgente.
batDelete	Cancella tutti i record della tabella attiva presenti anche nella tabella sorgente.
batUpdate	Aggiorna i record esistenti nella tabella attiva con record corrispondenti della tabella sorgente.

Ad esempio, con questo codice si aggiornano tutti i record della tabella attiva con record della tabella *Customer* che hanno gli stessi valori per i campi dell'indice attivo:

```
Table1->BatchMove("CUSTOMER.DB", batUpdate);
```

BatchMove restituisce il numero di record importati con successo.



Importando record con il modo *batCopy* si sovrascrivono i record esistenti. Per conservarli si ricorre invece a *batAppend*.

BatchMove esegue solo alcune delle operazioni batch supportate da BDE. Il componente *TBatchMove* dispone di ulteriori funzioni. Se è necessario spostare una grande quantità di dati tra tabelle, è preferibile utilizzare *TBatchMove* invece di chiamare un metodo *BatchMove*. Per informazioni sull'utilizzo di *TBatchMove*, consultare "Utilizzo di TBatchMove" a pagina 24-50.

Utilizzo di TQuery

TQuery rappresenta una singola istruzione Data Definition Language (DDL) o Data Manipulation Language (DML) (ad esempio, un comando SELECT, INSERT, DELETE, UPDATE, CREATE INDEX o ALTER TABLE). Il linguaggio nei comandi dipende dal server, ma di solito è conforme allo standard SQL-92 del linguaggio SQL. *TQuery* implementa tutte le funzionalità di base introdotte da *TDataSet*, come pure tutte le funzionalità speciali tipiche dei dataset di tipo query. Prima di analizzare le funzionalità esclusive di cui *TQuery* dispone, è bene prendere familiarità con le funzioni database fondamentali descritte in "I dataset", tra cui la sezione sui dataset di tipo query che inizia a pagina 22-43.

Siccome *TQuery* è un dataset per BDE, si deve di solito associarlo a un database e a una sessione. (L'unica eccezione è quando si utilizza *TQuery* per una query eterogenea). Il paragrafo "Associazione di un dataset a connessioni di database e di sessione" a pagina 24-3 descrive come si formano queste associazioni. Si specifica l'istruzione SQL della query impostando la proprietà *SQL*.

Un componente *TQuery* può accedere a dati in:

- Tabelle Paradox o dBASE, utilizzando Local SQL, che fa parte di BDE. Local SQL è un sottoinsieme delle specifiche SQL-92. Supporta la maggior parte del DML e una porzione di DDL sufficiente a funzionare con questi tipi di tabelle. Per i dettagli sulla sintassi SQL supportata, consultare la guida SQL locale, LOCALSQL.HLP..
- Database locali InterBase Server, utilizzando il motore InterBase. Per informazioni sul supporto della sintassi SQL standard e della sintassi estesa SQL-92 di InterBase, consultare *Language Reference* di InterBase.
- Database su server di database remoti come Oracle, Sybase, MS-SQL Server, Informix, DB2 e InterBase. Per accedere a un server remoto è necessario installare il driver SQL Link e appropriato e il software client (fornito dal produttore) specifico per il server di database. Viene accettata qualsiasi sintassi standard SQL supportata da questi server. Per informazioni sulla sintassi SQL, sulle sue limitazioni ed estensioni, fare riferimento alla documentazione del server in questione.

Creazione di query eterogenee

TQuery supporta query eterogenee verso più server o più tipi di tabella (ad esempio, dati da una tabella Oracle e da una tabella Paradox). Quando si esegue una query eterogenea, BDE analizza ed elabora la query utilizzando Local SQL e per questa ragione non è supportata la sintassi SQL specifica del server.

Per eseguire una query eterogenea, seguire questi passi:

1 Definire alias BDE distinti per ogni database cui si deve accedere nella query utilizzando lo strumento BDE BDE Administration o SQL Explorer.

2 Lasciare in bianco la proprietà *DatabaseName* di *TQuery*; i nomi dei database utilizzati saranno specificati nell'istruzione SQL.

3 Nella proprietà SQL, specificare l'istruzione SQL da eseguire. Far precedere ogni nome di tabella nell'istruzione con l'alias BDE del database della tabella, racchiuso tra segni di due punti. L'intero riferimento viene poi racchiuso tra virgolette.

4 Impostare i parametri della query nella proprietà *Params*.

5 Chiamare *Prepare* per preparare la query all' esecuzione prima di eseguirla per la prima volta.

6 Chiamare *Open* o *ExecSQL*, a seconda del tipo di query che si esegue.

Ad esempio, si supponga di definire un alias chiamato *Oracle1* per un database Oracle che ha una tabella CUSTOMER e *Sybase1* per un database Sybase che ha una tabella ORDERS. Una semplice query verso queste due tabelle sarebbe:

```
SELECT Customer.CustNo, Orders.OrderNo
FROM ":Oracle1:CUSTOMER"
  JOIN ":Sybase1:ORDERS"
    ON (Customer.CustNo = Orders.CustNo)
WHERE (Customer.CustNo = 1503)
```

Come alternativa all'utilizzo di un alias BDE per specificare il database in una query eterogenea, si può utilizzare un componente *TDatabase*. Configurare il *TDatabase* come normale per puntare al database, impostare il *TDatabase.DatabaseName* a un valore arbitrario ma univoco, quindi utilizzarlo nell'istruzione SQL al posto del nome dell'alias BDE.

Ottenimento di un set risultato modificabile

Per richiedere un set risultato che gli utenti possono editare in controlli data-aware, impostare a **true** la proprietà *RequestLive* di un componente query. L'impostazione di *RequestLive* a **true** non garantisce un set risultato "live" ma BDE fa il possibile per rispettare la richiesta. Le richieste di set risultato "live" sono soggette ad alcune limitazioni, a seconda che la query utilizzi il parser locale SQL, o un parser SQL del server.

- Le query in cui i nomi di tabella sono preceduti da un alias di database BDE (come nelle query eterogenee) e le query eseguite verso Paradox o dBASE, sono analizzate da BDE utilizzando Local SQL. Quando le query utilizzano il parser di Local SQL, BDE offre un supporto esteso per set risultato aggiornabili e "live" sia nelle tabelle singole che nelle query multitabella. Quando si utilizza Local SQL, una query su una singola tabella o vista restituisce un set risultato "live" se non contiene:
  - DISTINCT nella clausola SELECT
  - Join (interno, esterno, o UNION)
  - Funzioni di aggregazione con o senza clausole GROUP BY o HAVING
  - Tabelle o viste base non aggiornabili
  - Sottoquery

  - Clausole ORDER BY non basate su un indice
- Le query verso un server di database remoto vengono analizzate dal server. Se la proprietà *RequestLive* è impostata a **true**, l'istruzione SQL deve attenersi agli standard Local SQL, oltre a rispettare tutte le limitazioni imposte dal server, perché BDE deve utilizzarla per convogliare alla tabella i dati modificati. Una query su una singola tabella o vista restituisce un set risultato "live" se la query non contiene:
  - Una clausola DISTINCT nell'istruzione SELECT
  - Funzioni aggregate, con o senza clausole GROUP BY o HAVING
  - Riferimenti a più tabelle base o viste aggiornabili (concatenamenti)
  - Subquery che fanno riferimento alla tabella nella clausola FROM o altre tabelle

Se un'applicazione richiede e riceve un set risultato "live", la proprietà *CanModify* del componente query è impostata a **true**. La query, anche se restituisce un set risultato "live", potrebbe non essere in grado di aggiornare direttamente il set risultato nel caso contenesse campi collegati o se si commutassero gli indici prima di tentare l'aggiornamento. In presenza di queste condizioni, il set risultato andrebbe trattato come se fosse a sola lettura e andrebbe aggiornato di conseguenza.

Se un'applicazione richiede un set risultato "live", ma la sintassi dell'istruzione SELECT non lo consente, BDE restituisce

- Un set risultato a sola lettura nel caso di query rivolte a Paradox o a dBASE.
- Un codice di errore nel caso di query SQL rivolte a un server remoto.

Aggiornamento di set risultato a sola lettura

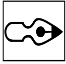
Le applicazioni possono aggiornare dati restituiti in un set risultato a sola lettura se utilizzano gli aggiornamenti in cache.

Se per gli aggiornamenti in cache si utilizza un dataset client, questo, o il provider associato può generare automaticamente lo SQL per effettuare gli aggiornamenti, a meno che la query non rappresenti più tabelle. In questo caso, è necessario indicare come applicare gli aggiornamenti:

- Se tutti gli aggiornamenti sono applicati a una sola tabella di database, è possibile indicare alla tabella in questione l'aggiornamento tramite un gestore di evento *OnGetTableName*.
- Se gli aggiornamenti vanno tenuti sotto stretto controllo, si può associare la query a un oggetto update (*TUpdateSQL*). Un provider utilizza automaticamente questo oggetto update per applicare gli aggiornamenti:

  1. Associare l'oggetto update alla query, applicando la proprietà *UpdateObject* della query all'oggetto *TUpdateSQL* che si utilizza.
  2. Impostare le proprietà *ModifySQL*, *InsertSQL* e *DeleteSQL* dell'oggetto update a istruzioni SQL che eseguono gli opportuni aggiornamenti dei dati della query.

Se si utilizza BDE per gli aggiornamenti in cache, è necessario utilizzare un oggetto update.



Per ulteriori informazioni sull'utilizzo di oggetti update, consultare "Aggiornamento di dataset mediante oggetti update" a pagina 24-41.

Utilizzo di TStoredProc

TStoredProc rappresenta una procedura registrata e implementa tutte le funzionalità di base introdotte da *TDataSet*, in aggiunta alla maggior parte delle funzioni speciali tipiche dei dataset di tipo procedura registrata. Prima di studiare le funzioni esclusive introdotte da *TStoredProc*, è bene impadronirsi delle funzionalità database comuni descritte in "I dataset",, compresa la sezione sui dataset di tipo procedura registrata che inizia a pagina pagina 22-52.

TStoredProc è un dataset per BDE, per cui va associato a un database e a una sessione. Il paragrafo "Associazione di un dataset a connessioni di database e di sessione" a pagina 24-3 spiega come formare queste associazioni. Una volta associato il dataset è a un database e a una sessione, è possibile collegarlo a una particolare procedura registrata impostando la proprietà *StoredProcName*.

TStoredProc differisce dagli altri dataset di tipo procedura registrata per queste ragioni:

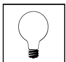
- Consente di controllare meglio la connessione dei parametri.
- Supporta le procedure registrate ridefinite Oracle.

Inglobamento dei parametri

Quando una procedura registrata viene preparata ed eseguita, i relativi parametri di input sono collegati automaticamente a parametri sul server.

TStoredProc consente di usare la proprietà *ParamBindMode* per specificare in che modo collegare i parametri ai parametri sul server. Per impostazione predefinita *ParamBindMode* è impostato a *pbByName*, cioè i parametri provenienti dal componente procedura registrata corrispondono per nome a del server. Questo è il metodo più facile di collegare i parametri.

Determinati server supportano anche i parametri di connessione secondo il valore ordinale, cioè secondo l'ordine in cui i parametri vengono visualizzati nella procedura registrata. In questo caso, è significativo l'ordine in cui si specificano i parametri nell'editor di raccolta dei parametri. Il primo parametro specificato corrisponde al primo parametro di input sul server, il secondo parametro al secondo parametro di input sul server, e così via. Se il server supporta la connessione di parametri secondo il valore ordinale, si può impostare *ParamBindMode* a *pbByNumber*.

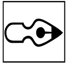


Se si desidera impostare *ParamBindMode* a *pbByNumber*, è necessario specificare i tipi di parametro corretti nell'ordine corretto. Nello SQL Explorer si può visualizzare il codice sorgente di una procedura registrata di un server per determinare l'ordine corretto e il tipo di parametri da specificare.

Operazioni con procedure registrate ridefinite di Oracle

I server Oracle consentono la ridefinizionedelle procedure registrate; le procedure ridefinite sono procedure diverse che hanno lo stesso nome. La proprietà *Overload* del componente procedura registrata permette a un'applicazione di specificare quale procedura eseguire.

Se *Overload* è zero (l'impostazione predefinita), si presume che non vi sia nessuna ridefinizione. Se *Overload* è uno (1), il componente procedura registrata esegue la prima procedura registrata che trova sul server Oracle contenente il nome ridefinito; se è due (2), esegue la seconda, e così via.



Le procedure registrate ridefinite accettano parametri di input e di output diversi. Per ulteriori informazioni, fare riferimento alla documentazione del server Oracle.

Collegamento ai database tramite TDatabase

Quando un'applicazione C++Builder utilizza Borland Database Engine (BDE) per collegarsi a un database, la connessione è incorporata in un componente *TDatabase*. Un componente database rappresenta la connessione a un singolo database, nel contesto di una sessione BDE.

TDatabase esegue molte delle attività di altri componenti di connessione ai database con i quali ha in comune molte proprietà , metodi ed eventi. Proprietà , metodi ed eventi comuni sono spiegati nel Capitolo 21, "Connessione ai database.".

In aggiunta a questo, *TDatabase* introduce molte funzioni specifiche di BDE, che vengono descritte qui di seguito.

Associazione di un componente database a una sessione

Tutti i componenti database devono essere associati a una sessione BDE. Per stabilire l'associazione si ricorre a *SessionName*. Quando si crea un componente database in fase di progetto, *SessionName* è impostato a "Default", il che significa che è associato al componente session predefinito che è referenziato dalla variabile globale *Session*.

Le applicazioni multithread o reentrant BDE possono richiedere più di una sessione. Se è necessario utilizzare sessioni multiple, bisogna aggiungere componenti *TSession* per ciascuna sessione. Poi si associata il dataset a un componente session impostando la proprietà *SessionName* alla proprietà *SessionName* di un componente session.

In esecuzione, si accede al componente session cui il database è associato leggendo la proprietà *Session*. Se *SessionName* è vuoto o ha il valore "Default", allora la proprietà *Session* fa riferimento alla stessa istanza *TSession* referenziata dalla variabile globale *Session*. *Session* consente alle applicazioni di accedere alle proprietà , ai metodi e agli eventi del componente session superiore di un componente database senza conoscere il nome effettivo della sessione.

Per ulteriori informazioni sulle sessioni BDE, vedere "Gestione delle sessioni di database" a pagina 24-17.

Se si utilizza un componente database implicito, la sessione relativa per quel componente database è quella specificata dalla proprietà *SessionName* del dataset.

Interazioni tra il componente Session e i database

Di solito, le proprietà del componente session determinano i comportamenti globali e predefiniti applicati a tutti i componenti database impliciti creati in fase di esecuzione. Ad esempio, la proprietà *KeepConnections* della sessione controllante determina se la connessione con un database rimane attiva anche quando sono chiusi i dataset associati (impostazione predefinita), o se i collegamenti vengono interrotti quando tutti i suoi dataset sono chiusi. Similmente, l'evento predefinito *OnPassword* di una sessione garantisce che quando un'applicazione tenta di collegarsi a un database su un server che richiede una password, appare una finestra di dialogo standard con la richiesta di indicare la password.

L'applicazione dei metodi della sessione è piuttosto diversa. I metodi di *TSession* influiscono su tutti i componenti database, sia che vengano creati esplicitamente oppure istanziati implicitamente da un dataset. Ad esempio, il metodo di sessione *DropConnections* chiude tutti i dataset che appartengono ai componenti database di una sessione, poi interrompe tutti i collegamenti con il database, anche se la proprietà *KeepConnection* per singoli componenti database è **true**.

I metodi dei componenti database si applicano solo ai dataset associati a un determinato componente database. Ad esempio, si supponga che il componente database *Database1* sia associato alla sessione predefinita. *Database1->CloseDataSets()* chiude solo i dataset associati a *Database1*. I dataset aperti appartenenti ad altri componenti database all'interno della sessione predefinita rimarranno aperti.

Identificazione del database

AliasName e *DriverName* sono proprietà reciprocamente esclusive che identificano il server di database cui si collega il componente *TDatabase*.

- *AliasName* specifica il nome di un alias BDE esistente da usare con il componente database. L'alias appare in liste a discesa successive dei componenti dataset consentendo la connessione al componente database desiderato. Se si specifica *AliasName* di un componente database, l'eventuale valore assegnato a *DriverName* viene cancellato perché il nome del driver fa sempre parte di un alias BDE.

  Gli alias BDE si creano e si modificano con Database Explorer o con il programma di utilità BDE Administration. Per ulteriori informazioni sulla creazione e la manutenzione degli alias BDE, consultare la relativa documentazione in linea.

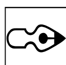
- *DriverName* è il nome di un driver BDE. Un nome di driver è un parametro in un alias BDE , ma si può specificare il nome del driver invece dell'alias quando si crea l'alias locale BDE di un componente database utilizzando la proprietà *DatabaseName*. Specificando *DriverName*, l'eventuale valore precedentemente assegnato ad *AliasName* viene cancellato per evitare potenziali conflitti tra il nome del driver che si specifica e quello che fa parte dell'alias BDE identificato in *AliasName*.

DatabaseName consente di fornire il nome di una connessione a un database. Il nome fornito si aggiunge ad *AliasName* o a *DriverName* ed è locale all'applicazione. *DatabaseName* può essere un alias BDE, o, nel caso di file Paradox e dBASE, il nome completo del percorso. Come *AliasName*, *DatabaseName* viene visualizzato in liste a

discesa di componenti dataset in modo da consentire la connessione ai componenti database.

In progettazione, per specificare un alias BDE, si assegna un driver BDE o si crea un alias locale BDE, facendo doppio clic su un componente database e richiamando così Database Properties editor.

È possibile immettere un *DatabaseName* nella casella di modifica Name nell'editor delle proprietà . Si può immettere un nome di alias BDE esistente nella casella combinata nome Alias della proprietà *Alias* oppure selezionare tra gli alias esistenti nella lista a discesa. La casella combinata Driver name permette di immettere il nome di un driver BDE esistente per la proprietà *DriverName*, o altrimenti di scegliere tra i nomi dei driver esistenti nella lista a discesa.



L'editor Database Properties permette anche di visualizzare e di impostare i parametri di connessione BDE e di impostare gli stati delle proprietà *LoginPrompt* e *KeepConnection*. Per informazioni sui parametri di connessione, consultare "Impostazione dei parametri degli alias BDE" qui di seguito. Per informazioni su *LoginPrompt*, consultare "Controllo del login al server" a pagina 21-4. Per informazioni su *KeepConnection* consultare "Apertura di una connessione utilizzando TDatabase" a pagina 24-16.

Impostazione dei parametri degli alias BDE

In progettazione è possibile creare o modificare i parametri di connessione in tre modi:

- Utilizzare il programma di utilità Database Explorer o BDE Administration per creare o modificare alias BDE, tra cui i parametri. Per ulteriori informazioni su questi programmi di utilità , consultare i relativi file della Guida in linea.
- Richiamare l'editor String List facendo doppio clic sulla proprietà *Params* nell'Object Inspector.
- Richiamare l'editor Database Properties facendo doppio clic su un componente database in un modulo dati o in una scheda.

Con tutti questi metodi si può modificare la proprietà *Params* del componente database. *Params* è una lista di stringhe che contiene i parametri di connessione al database relativi all'alias BDE associato a un componente database. Alcuni parametri di connessione tipici comprendono percorso, nome server, dimensione della cache di schema, driver di linguaggio e modalità della query SQL.

Quando si richiama il Database Properties editor, i parametri dell'alias BDE non sono visibili. Per vedere le impostazioni attive, fare clic su Defaults. I parametri attivi vengono visualizzati nel riquadro di nota Parameter overrides. È possibile modificare le voci esistenti o aggiungerne di nuove. Per cancellare i parametri esistenti, fare clic su Clear. Le modifiche apportate agiscono quando si fa clic su OK.

In fase di esecuzione, un'applicazione può impostare i parametri alias solo modificando direttamente la proprietà *Params*. Per ulteriori informazioni su parametri specifici per l'impiego dei di driver SQL Links con BDE, consultare il file della guida in linea SQL Links.

Apertura di una connessione utilizzando TDatabase

Come avviene per tutti i componenti di connessione ai database, per collegarsi a un database mediante *TDatabase* si imposta la proprietà *Connected* a **true** o si chiama il metodo *Open*. La procedura è descritta in "Connessione a un server di database" a pagina 21-3. Una connessione, una volta che è stabilita, rimane attiva fino a quando almeno un dataset è attivo. Quando non ci sono più dataset attivi, la connessione viene interrotta, a meno che non sia **true** la proprietà *KeepConnection* del componente database.

Quando da un'applicazione ci si collega a un server di database remoto, l'applicazione stabilisce la connessione tramite BDE e il driver Borland SQL Links. (BDE può anche comunicare con un driver ODBC che bisogna fornire). Prima di stabilire la connessione bisogna configurare il driver SQL Links o ODBC dell'applicazione. I parametri SQL Links e ODBC sono memorizzati nella proprietà *Params* del componente database. Per informazioni su parametri SQL Links, consultare SQL Links User's Guide. Per modificare la proprietà *Params*, consultare "Impostazione dei parametri degli alias BDE" a pagina 24-15.

Operazioni con i protocolli di rete

Per configurare il driver SQL Links o ODBC appropriato, può essere necessario specificare il protocollo di rete utilizzato dal server, come SPX/IPX o TCP/IP, a seconda delle opzioni di configurazione del driver. Nella maggior parte dei casi, la configurazione del protocollo di rete si gestisce tramite il software di installazione client del server. Nel caso di ODBC potrebbe anche essere necessario controllare l'installazione del driver tramite il driver manager di ODBC.

Stabilire il collegamento tra client e server a volte è problematico. La seguente lista delle anomalie di funzionamento si rivela utile se si incontrano difficoltà :

- La connessione dal lato client del server è configurata correttamente?
- I DLL dei driver di connessione e di database si trovano nel percorso di ricerca?
- Se si utilizza TCP/IP:
  - Il software di comunicazione TCP/IP è stato installato? È stato installato il WINSOCK.DLL corretto?
  - L'indirizzo IP del server è registrato nel file HOSTS del client?
  - Il Domain Name Services (DNS) è configurato correttamente?
  - È possibile eseguire il ping sul server?

Per maggiori informazioni sulla ricerca delle anomalie, consultare SQL Links User's Guide e la documentazione del server.

Uso di ODBC

Un'applicazione può utilizzare delle sorgenti dati ODBC (ad esempio, Btrieve). Una connessione tramite un driver ODBC richiede

- Un driver ODBC acquistabile separatamente.
- Il Microsoft ODBC Driver Manager.

- Il programma di utilità BDE Administration.

Per configurare un alias BDE di una connessione driver ODBC, si utilizza il programma di utilità BDE Administration. Per ulteriori informazioni, consultare la guida in linea del programma di utilità BDE Administration.

Impiego di componenti database nei moduli dati

I componenti database si inseriscono in tutta sicurezza nei moduli dati. Tuttavia, se si mette nell’Object Repository un modulo dati contenente un componente database e si desidera che altri utenti siano in grado di ereditare da esso, è necessario impostare a **true** la proprietà *HandleShared* del componente database per evitare conflitti di spazio di nome globali.

Gestione delle sessioni di database

Le connessioni di database di un’applicazione BDE, i driver, i cursori, le query, e così via sono gestite all’interno del contesto di una o più sessioni BDE. Le sessioni isolano un gruppo di operazioni di accesso al database, come i collegamenti, senza la necessità di avviare un’altra istanza dell’applicazione.

Tutte le applicazioni database BDE includono automaticamente un componente session predefinito, chiamato *Session*, che comprende la sessione BDE predefinita. I componenti database aggiunti all’applicazione vengono associati automaticamente alla sessione predefinita (osservare che il suo *SessionName* è ”Default”). La sessione predefinita fornisce il controllo totale di tutti i componenti database non associati a un’altra sessione, sia che siano impliciti (creati dalla sessione in esecuzione all’apertura di un dataset non associato a un componente database creato) o stabili (creati esplicitamente dall’applicazione). In fase di progetto, la sessione predefinita non è visibile nel modulo dati o nella scheda, ma in fase di esecuzione è possibile accedere alle sue proprietà e ai suoi metodi nel codice.

Per utilizzare la sessione predefinita, non si deve scrivere alcun codice, a meno che l’applicazione non debba

- Attivare esplicitamente o disattivare una sessione, attivando o disattivando la capacità di aprirsi dei database della sessione.
- Modificare le proprietà della sessione, come la specifica delle proprietà predefinite dei componenti database generati implicitamente.
- Eseguire i metodi di una sessione, come la gestione dei collegamenti al database (ad esempio aprendo e chiudendo la connessione in risposta ad azioni dell’utente).
- Rispondere a eventi di sessione, come avviene quando l’applicazione tenta di accedere a una tabella Paradox o dBASE protetta da password.
- Indicare la directory secondo le specifiche Paradox, come la proprietà *NetFileDir* per accedere a tabelle Paradox in rete e la proprietà *PrivateDir* sullo hard disk locale per per migliorare le prestazioni.
- Gestire gli alias BDE che descrivono possibili configurazioni di connessione al database, relativamente a database e a dataset che utilizzano la sessione.

Sia che si aggiungano componenti database a un'applicazione in fase di progettazione, sia che si creino dinamicamente in fase di esecuzione, essi vengono associati automaticamente alla sessione predefinita, a meno che non li si assegni specificatamente a una sessione diversa. Se si apre un dataset non associato a un componente database, Delphi automaticamente

- Crea per esso un componente database in fase di esecuzione.
- Associa il componente database alla sessione predefinita.
- Inizializza alcune proprietà chiave del componente database in base alle proprietà della sessione predefinita. Una delle più importanti è *KeepConnections*, che determina quando un'applicazione deve mantenere attivo o chiudere i collegamenti.

La sessione predefinita fornisce un insieme ampiamente applicabile di impostazioni predefinite utilizzabili così come sono nella maggior parte delle applicazioni. È necessario solo associare un componente database a una sessione esplicitamente chiamata, se il componente esegue una query simultanea verso un database già aperto dalla sessione predefinita. In questo caso, ogni query simultanea va eseguita in una propria sessione. Anche le applicazioni database multithread richiedono sessioni multiple, in cui ogni thread ha la propria sessione.

Le applicazioni possono creare ulteriori componenti session in base alle necessità . Le applicazioni database BDE includono automaticamente un componente session list, chiamato *Sessions*, utilizzabile per gestire tutti i componenti session. Per ulteriori informazioni sulla gestione di sessioni multiple, consultare "Gestione di sessioni multiple" a pagina 24-29.

I componenti session si possono mettere senza problemi anche nei moduli dati. Se si mette un modulo dati che contiene uno o più componenti session nell'Object Repository, tuttavia, bisogna assicurarsi che la proprietà *AutoSessionName* sia **true** per evitare conflitti di nome quando gli utenti ereditano da esso.

Attivazione di una sessione

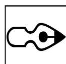
Active è una proprietà Boolean che determina se il database e i componenti dataset associati a una sessione sono aperti. La proprietà può servire per leggere lo stato corrente delle connessioni del database e del dataset di una sessione o per modificarlo. Se *Active* è **false** (l'impostazione predefinita), tutti i database e tutti i dataset associati alla sessione sono chiusi. Se è **true**, database e dataset sono aperti.

Una sessione viene attivata quando la si crea e, successivamente, ogni volta che la sua proprietà *Active* viene modificata a **true** da **false** (ad esempio, quando un database o un dataset è associato con una sessione aperta e non ci sono altri database o dataset aperti). Impostando *Active* a **true** si innesca l'evento *OnStartup* della sessione, si registrano le posizioni di directory Paradox con BDE e si registra la proprietà *ConfigMode*, che determina quali alias BDE sono disponibili all'interno della sessione. È possibile scrivere un gestore di evento *OnStartup* per inizializzare le proprietà *NetFileDir*, *PrivateDir* e *ConfigMode* prima che esse siano registrate con BDE, o per eseguire altre specifiche attività di avvio. Per informazioni sulle proprietà *NetFileDir* e *PrivateDir*, consultare "Specifica delle ubicazioni delle directory di Paradox" a

pagina 24-25. Per informazioni su ConfigMode, consultare "Operazioni con gli alias BDE" a pagina 24-25.

Una volta attivata la sessione, è possibile aprire le sue connessioni con il database chiamando il metodo *OpenDatabase*.

Nel caso di componenti session inseriti in un modulo dati o in una scheda, impostando *Active* a **false** si chiudono i database o i dataset aperti. In fase di esecuzione, la chiusura dei database e dei dataset attiva gli eventi associati.



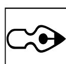
In fase di progettazione, non è consentito impostare *Active* a **false** per la sessione predefinita. Sebbene sia possibile chiudere la sessione predefinita in fase di esecuzione, è sconsigliabile farlo.

In fase di esecuzione, si possono anche utilizzare i metodi *Open* e *Close* di una sessione per attivare o disattivare sessioni diverse dalla sessione predefinita. Ad esempio, con quest'unica riga di codice si chiudono tutti i database e i dataset aperti di una sessione:

```
Session1->Close();
```

Questo codice imposta la proprietà *Active* di Session1 a **false**. Quando la proprietà *Active* di una sessione è **false**, un ulteriore tentativo dell'applicazione di aprire un database o un dataset reimposta *Active* a **true** e chiama il gestore di evento *OnStartup* della sessione, se esiste. È possibile anche codificare esplicitamente la riattivazione della in fase di esecuzione. Il seguente codice riattiva *Session1*:

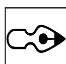
```
Session1->Open();
```



Quando una sessione è attiva si possono aprire e chiudere i singoli collegamenti al database. Per ulteriori informazioni, consultare "Chiusura delle connessioni con il database" a pagina 24-20.

Specifica del comportamento predefinito della connessione al database

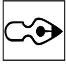
KeepConnections fornisce il valore predefinito della proprietà *KeepConnection* dei componenti database impliciti creati in fase di esecuzione. *KeepConnection* specifica che cosa accade a una connessione di database stabilita per un componente database quando tutti i suoi dataset sono chiusi. Se è **true** (l'impostazione predefinita), si stabilisce una connessione costante o *stabile* anche se non ci sono dataset attivi. Quando è **false**, la connessione viene interrotta non appena tutti i suoi dataset sono chiusi.



La persistenza della connessione di un componente database inserito esplicitamente in un modulo dati o in una scheda è controllata dalla sua proprietà *KeepConnection*. Se impostata diversamente, *KeepConnection* del componente database prevale sempre sulla proprietà *KeepConnections* della sessione. Per ulteriori informazioni sul controllo delle singole connessioni database all'interno di una sessione, consultare "Gestione delle connessioni con il database" a pagina 24-20.

KeepConnections dovrebbe essere impostato a **true** nelle applicazioni che aprono e chiudono frequentemente tutti i dataset associati a un database situato su un server remoto. Questa impostazione riduce il traffico di rete e velocizza l'accesso ai dati, in quanto significa che nel corso della sessione la connessione viene aperta e chiusa solo una volta. Altrimenti, l'applicazione ogni volta che chiude o ristabilisce la

connessione, deve affrontare l'incombenza aggiuntiva della connessione o dello scollegamento con il database.



Anche quando in una sessione *KeepConnections* è **true**, chiamando il metodo *DropConnections* è possibile chiudere e liberare i collegamenti database non attivi relativamente a tutti i componenti database impliciti. Per ulteriori informazioni su *DropConnections*, consultare "Rilascio delle connessioni database non attive" a pagina 24-21.

Gestione delle connessioni con il database

È possibile utilizzare un componente session per gestire i collegamenti con un database. Il componente session contiene le proprietà e i metodi che utilizzabili per

- Aprire i collegamenti con il database.
- Chiudere i collegamenti con il database.
- Chiudere e disimpegnare tutti i collegamenti con il database non attivi.
- Individuare collegamenti specifici con il database.
- Iterare attraverso tutte le connessioni con il database aperte.

Apertura delle connessioni con il database

Per aprire una connessione database all'interno di una sessione, si chiama il metodo *OpenDatabase*. *OpenDatabase* accetta un parametro, il nome del database da aprire. Il nome è un alias BDE, oppure il nome di un componente database. Nel caso di Paradox o di dBASE, il nome può anche essere il nome del percorso per esteso. Ad esempio, la seguente istruzione utilizza la sessione predefinita e tenta di stabilire la connessione con il database puntato dall'alias BCDEMOS:

```
TDatabase *BCDemosDatabase = Session->OpenDatabase("BCDEMOS");
```

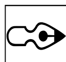
OpenDatabase attiva la sessione se questa non è già attiva e poi verifica se il nome del database specificato corrisponde alla proprietà *DatabaseName* di qualsiasi componente database della sessione. Se il nome non corrisponde a un componente database esistente, *OpenDatabase* ne crea uno temporaneo utilizzando il nome specificato. Infine, *OpenDatabase* chiama il metodo *Open* del componente database per la connessione al server. Ogni chiamata a *OpenDatabase* fa aumentare di un'unità il numero di riferimento del database. Il database rimane aperto fino a quando il numero di riferimento è maggiore di 0.

Chiusura delle connessioni con il database

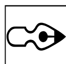
Per chiudere una singolo collegamento con un database, si chiama il metodo *CloseDatabase*. Quando si chiama *CloseDatabase*, il numero di riferimento del database, che viene incrementato ogni volta che si chiama *OpenDatabase*, diminuisce di un'unità . Quando il numero di riferimento è 0, il database viene chiuso. *CloseDatabase* accetta un parametro, il database da chiudere. Se il database è stato aperto utilizzando il metodo *OpenDatabase*, questo parametro può essere impostato al valore restituito da *OpenDatabase*.

```
Session->CloseDatabase(BCDemosDatabase);
```

Se il nome del database specificato è associato a un componente database temporaneo (implicito) e la proprietà *KeepConnections* della sessione è **false**, il componente database viene liberato, e la connessione viene effettivamente chiusa.



Se *KeepConnections* è **false** i componenti database temporanei vengono chiusi e liberati automaticamente quando si chiude l'ultimo dataset associato al componente database. Un'applicazione può sempre chiamare *CloseDatabase* in anticipo per forzare la chiusura. Per liberare dei componenti database temporanei quando *KeepConnections* è **true**, si chiamano il metodo *Close* del componente database, e poi il metodo *DropConnections* della sessione.



La chiamata di *CloseDatabase* per un componente database costante in realtà non chiude la connessione. Allo scopo, bisogna chiamare direttamente il metodo *Close* del componente database.

Ci sono due modi per chiudere tutte le connessioni database all'interno della sessione:

- Impostare la proprietà *Active* della sessione a **false**.
- Chiamare il metodo *Close* della sessione.

Quando *Active* ha il valore **false**, C++Builder chiama automaticamente il metodo *Close*. *Close* si scollega da tutti i database attivi, liberando i componenti database temporanei e chiamando ogni metodo *Close* del componente database stabile. Infine, *Close* assegna il valore **NULL** al gestore BDE della sessione.

Rilascio delle connessioni database non attive

Se la proprietà *KeepConnections* di una sessione è **true** (l'impostazione predefinita), allora le connessioni database dei componenti database temporanei rimangono attive anche se sono chiusi tutti i dataset utilizzati dal componente. È possibile eliminare i collegamenti in questione e liberare tutti i componenti database temporanei non attivi di una sessione chiamando il metodo *DropConnections*. Ad esempio, il seguente codice libera tutti i componenti database non attivi e temporanei della sessione predefinita:

```
Session->DropConnections();
```

I componenti database temporanei che hanno uno o più dataset attivi non vengono rilasciati o liberato da questa chiamata. Per liberare questi componenti, chiamare *Close*.

Ricerca di una connessione con un database

Per determinare se un componente database specificato è già associato a una sessione si utilizza il il metodo *FindDatabase*. *FindDatabase* accetta un parametro, il nome del database da cercare, che può essere l'alias BDE o il nome del componente database. Per Paradox o per dBASE, può anche essere il nome del percorso per esteso.

FindDatabase restituisce il componente database se trova una corrispondenza. Altrimenti restituisce **NULL**.

Il seguente codice cerca nella sessione predefinita un componente database utilizzando l'alias BCDEMOS e, se non lo trova, ne crea uno e lo apre:

```
TDatabase *DB = Session->FindDatabase("BCDEMOS");
```

```
if ( !DB )                            // Database does not exist for session so
  DB = Session->OpenDatabase("BCDEMOS"); // create and open it
if (DB && DB->Connected)
{
  if (!DB->InTransaction)
  {
    DB->StartTransaction();
    ⋮
  }
}
```

Iterazione tra i componenti database di una sessione

È possibile utilizzare due proprietà del componente sessione, *Databases* e *DatabaseCount*, per iterare attraverso tutti i componenti database attivi associati a una sessione.

Databases è un array di tutti i componenti database attivi associati a una sessione. *DatabaseCount* è il numero di database dell’array. Mano a mano che i collegamenti si aprono e si chiudono nel corso della sessione, i valori di *Databases* e di *DatabaseCount* cambiano. Ad esempio, se la proprietà *KeepConnections* una sessione è **false** e in fase di esecuzione tutti i componenti database vengono creati in base alle necessità , ogni volta che un certo database viene aperto, *DatabaseCount* viene aumentato di un’unità . Ogni volta che un certo database viene chiuso, *DatabaseCount* diminuisce da uno. Se *DatabaseCount* è zero, la sessione non contiene componenti database attivi.

In questo esempio viene impostata a **true** la proprietà *KeepConnection* di ciascun database attivo nella sessione predefinita:

```
if (Session->DatabaseCount > 0)
  for (int MaxDbCount = 0; MaxDbCount < Session->DatabaseCount; MaxDbCount++)
    Session->Databases[MaxDbCount]->KeepConnection = true;
```

Operazioni con tabelle Paradox e dBASE protette da password

Un componente session può memorizzare le password delle tabelle Paradox e dBASE protette da password. Una volta aggiunta una password alla sessione, l’applicazione in grado di aprire le tabelle che la password protegge. Quando si toglie la password dalla sessione, l’applicazione non può aprire tabelle che utilizzano la password fino a quando questa non viene nuovamente aggiunta.

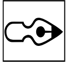
Uso del metodo AddPassword

Il metodo *AddPassword* è un modo facoltativo perché un’applicazione fornisca una password per una sessione prima dell’apertura di una tabella crittografata Paradox o dBASE alla quale si può accedere solo indicando la password. Se non si aggiunge la password alla sessione, una finestra di dialogo richiede una password all’utente quando l’applicazione tenta di aprire una tabella protetta da password.

AddPassword accetta un parametro contenente la password da utilizzare, una stringa. È possibile chiamare *AddPassword* ogni qual volta è necessario per aggiungere (una alla volta) le password che consentono di accedere alle tabelle protette con varie password.

```
AnsiString PassWrd;
```

```
PassWrd = InputBox("Enter password", "Password:", "");
Session->AddPassword(PassWrd);
try
{
  Table1->Open();
}
catch(...)
{
  ShowMessage("Could not open table!");
  Application->Terminate();
}
```



L'utilizzo della funzione *InputBox*, sopra, ha solo scopi dimostrativi. Nelle applicazioni reali, si ricorre alle funzionalità di immissione della password che nascondono i singoli caratteri durante l'immissione, come la funzione *PasswordDialog* o una scheda custom.

Il pulsante Add della finestra di dialogo di *PasswordDialog* agisce proprio come il metodo *AddPassword*.

```
if (PasswordDlg(Session))
  Table1->Open();
else
  ShowMessage("No password given, could not open table!");
```

Utilizzo dei metodi RemovePassword e RemoveAllPasswords

RemovePassword cancella dalla memoria una password precedentemente inserita. *RemovePassword* accetta un parametro contenente la password da cancellare.

```
Session->RemovePassword("secret");
```

RemoveAllPasswords cancella dalla memoria tutte le password precedentemente aggiunte.

```
Session->RemoveAllPasswords();
```

Utilizzo del metodo GetPassword e dell'evento OnPassword

L'evento *OnPassword* permette di controllare in che modo l'applicazione fornisce le password quando vengono richieste quelle relative alle tabelle Paradox e dBASE. Se si desidera ridefinire il comportamento di gestione della password predefinita bisogna fornire il gestore dell'evento *OnPassword*. Quando non si fornisce il gestore, C++Builder presenta una finestra di dialogo predefinita per l'immissione di una password ma non è previsto nessun comportamento particolare-o il tentativo di aprire la tabella riesce, oppure viene sollevata un'eccezione.

Se si fornisce il gestore dell'evento *OnPassword*, questo deve fare due cose: chiamare il metodo *AddPassword* e impostare il parametro *Continue* del gestore di evento a **true**. Il metodo *AddPassword* passa alla sessione una stringa da utilizzare come password della tabella. Il parametro *Continue* indica a C++Builder che non sono necessarie ulteriori richieste di password per questo tentativo di aprire la tabella. Il valore predefinito di *Continue* è **false** per cui bisogna dargli esplicitamente il valore **true**. Se *Continue* è **false**, terminata l'esecuzione del gestore di evento, riparte un evento *OnPassword* -anche se mediante *AddPassword* è stata passata la password valida. Se *Continue* è **true**, dopo l'esecuzione del gestore di evento e se la stringa passata con

AddPassword non è la password valida, il tentativo di aprire la tabella non riesce e viene sollevata un'eccezione.

OnPassword può essere innescato da due circostanze. Il primo è il tentativo di aprire una tabella protetta da password (dBASE o Paradox) quando alla sessione non è stata già fornita la password valida. (in caso contrario, l'evento *OnPassword* non si verifica).

L'altra circostanza è una chiamata al metodo *GetPassword*. *GetPassword* genera un evento *OnPassword* oppure, se la sessione non ha un gestore di evento *OnPassword*, visualizza una finestra di dialogo Default password. Restituisce **true** se il gestore di evento *OnPassword* o la finestra di dialogo predefinita hanno aggiunto una password alla sessione, e **false** se non è stata indicata nessuna stringa.

Nel seguente esempio, il metodo *Password* è indicato come gestore di evento *OnPassword* della sessione predefinita, assegnandolo alla proprietà *OnPassword* dell'oggetto globale *Session.*

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Session->OnPassword = Password;
}
```

Nel metodo *Password*, la funzione *InputBox* chiede la password all'utente. Il metodo *AddPassword* poi fornisce da programma alla sessione la password immessa nella finestra di dialogo.

```
void __fastcall TForm1::Password(TObject *Sender, bool &Continue)
{
  AnsiString PassWrd = InputBox("Enter password", "Password:", "");
  Session->AddPassword(PassWrd);
  Continue = (PassWrd > "");
}
```

L'evento *OnPassword* (e pertanto il gestore di evento *Password*) viene innescato da un tentativo di aprire una tabella protetta da password, come dimostrato qui di seguito. Sebbene il gestore di evento *OnPassword* chieda all'utente di fornire la password, il tentativo di aprire la tabella può fallire nel caso si immetta una password non valida o se qualcos'altro non va a buon fine.

```
void __fastcall TForm1::OpenTableBtnClick(TObject *Sender)
{
  try
  {
    // this line triggers the OnPassword event
    Table1->Open();
  }
  // exception if cannot open table
  catch(...)
  {
    ShowMessage("Could not open table!");
    Application->Terminate();
  }
}
```

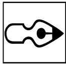
Specifica delle ubicazioni delle directory di Paradox

Due proprietà del componente sessione, *NetFileDir* e *PrivateDir*, riguardano specificamente le applicazioni che lavorano con tabelle Paradox.

NetFileDir specifica la directory che contiene PDOXUSRS.NET, il file di controllo di rete di Paradox. Questo file governa la condivisione delle tabelle Paradox sulle unità di rete. Tutte le applicazioni che devono condividere delle tabelle Paradox devono specificare la stessa directory del file di controllo di rete (di solito una directory su un file server di rete). Per un certo alias di database, C++Builder deriva il valore per *NetFileDir* dal file di configurazione di Borland Database Engine (BDE). Se si imposta personalmente *NetFileDir*, il valore fornito prevale sull'impostazione di configurazione di BDE, per cui bisogna assicurarsi di convalidare il nuovo valore.

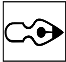
In fase di progetto, si può indicare il valore di *NetFileDir* tramite l'Object Inspector. Si può anche impostare o modificare *NetFileDir* nel codice, in fase di esecuzione. Il seguente codice impostare *NetFileDir* per la sessione predefinita all'ubicazione della directory da dove l'applicazione viene eseguita:

```
Session->NetFileDir = ExtractFilePath(ParamStr(0));
```



NetFileDir può essere modificato solo quando in un'applicazione non ci sono file Paradox aperti. Se si modifica *NetFileDir* in esecuzione, bisogna verificare che esso punti una directory di rete valida che sia condivisa dagli utenti di rete.

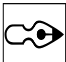
PrivateDir specifica la directory dove vengono registrati i i file temporanei di elaborazione delle tabelle, come quelli che BDE genera per gestire le istruzioni SQL locali. Se non viene specificato il valore della proprietà *PrivateDir*, BDE nel momento in cui viene inizializzato utilizza automaticamente la directory corrente. Se l'applicazione viene eseguita direttamente da un file server di rete, è possibile migliorare in esecuzione la performance dell'applicazione impostando *PrivateDir* su un disco rigido locale di un utente prima di aprire il database.



Non impostare *PrivateDir* in fase di progettazione per poi aprire la sessione nell'IDE. Così facendo, viene generato l'errore Directory is busy quando l'applicazione viene eseguita a partire è dall'IDE.

Il seguente codice modifica l'impostazione della proprietà *PrivateDir* della sessione predefinita sulla directory C: \TEMP di un utente:

```
Session->PrivateDir = "C:\\TEMP";
```



Non impostare *PrivateDir* sulla directory principale di un drive. Bisogna sempre specificare una subdirectory.

Operazioni con gli alias BDE

Ogni componente database associato a una sessione ha un alias BDE (benché il nome del percorso per esteso possa facoltativamente essere sostituito da un alias quando si accede a tabelle Paradox e dBASE). Una sessione quando è attiva può creare, modificare e cancellare gli alias.

Il metodo *AddAlias* crea un nuovo alias BDE per un server di database SQL. *AddAlias* prende tre parametri: una stringa che contiene un nome per l'alias, una stringa che specifica il driver SQL Links da utilizzare e una stringa lista popolata di parametri

per l'alias. Ad esempio, le seguenti istruzioni utilizzano *AddAlias* per aggiungere un nuovo alias per l'accesso a un server InterBase alla sessione predefinita:

```
TStringList *AliasParams = new TStringList();
try
{
  AliasParams->Add("OPEN MODE=READ");
  AliasParams->Add("USER NAME=TOMSTOPPARD");
  AliasParams->Add("SERVER NAME=ANIMALS:/CATS/PEDIGREE.GDB");
  Session->AddAlias("CATS", "INTRBASE", AliasParams);
  ⋮
}
catch (...)
{
  delete AliasParams;
  throw;
}
delete AliasParams;
```

AddStandardAlias crea un nuovo alias BDE per le tabelle Paradox, dBASE o ASCII. *AddStandardAlias* accetta tre parametri di stringa: il nome dell'alias, il percorso completo della tabella Paradox o dBASE cui si vuole accedere e il nome del driver predefinito driver da usare quando si cerca di aprire una tabella priva di estensione. Ad esempio, la seguente istruzione utilizza *AddStandardAlias* per creare un nuovo alias per accedere a una tabella Paradox:

```
Session->AddStandardAlias("MYBCDEMOS", "C:\\TESTING\\DEMOS\\", "Paradox");
```

Quando si aggiunge un alias a una sessione, BDE registra una copia dell'alias in memoria, dove è disponibile per la sessione in corso e per le altre sessioni che includano il valore *cfmPersistent* nella proprietà *ConfigMode*. *ConfigMode* è un set che descrive quali tipi di alias possono essere utilizzati dai database della sessione. L'impostazione predefinita è *cmAll*, che equivale al set [*cfmVirtual*, *cfmPersistent*, *cfmSession*]. Se *ConfigMode* è *cmAll*, la sessione vede tutti gli alias creati al suo interno (*cfmSession*), tutti gli alias nel file di configurazione BDE sul sistema dell'utente (*cfmPersistent*) e tutti gli alias che BDE conserva in memoria (*cfmVirtual*). È possibile modificare *ConfigMode* per limitare quali alias BDE i database in una sessione possono utilizzare. Ad esempio, l'impostazione di *ConfigMode* a *cfmSession* fa vedere alla sessione solo gli alias creati all'interno della sessione. Tutti gli altri alias presenti nel file di configurazione BDE e in memoria non sono disponibili.

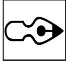
Per mettere un alias appena creato a disposizione di tutte le sessioni e di altre applicazioni, si ricorre al metodo *SaveConfigFile* della sessione. *SaveConfigFile* scrive alias in memoria sul file di configurazione BDE, dove altre applicazioni BDE capaci di funzionare con BDE possono leggerli e utilizzarli.

Dopo avere creato un alias, per modificarne i parametri si ricorre a *ModifyAlias*. *ModifyAlias* accetta due parametri: il nome dell'alias da modificare e una lista di stringhe contenente i parametri da cambiare e i rispettivi valori. Ad esempio, con le seguenti istruzioni *ModifyAlias* cambia in READ/WRITE il parametro OPEN MODE dell'alias CATS facendolo diventare nella sessione predefinita:

```
TStringList *List = new TStringList();
List->Clear();
List->Add("OPEN MODE=READ/WRITE");
```

```
Session->ModifyAlias("CATS", List);
delete List;
```

Per cancellare un alias precedentemente creato nel corso di una sessione, si ricorre al metodo *DeleteAlias*. *DeleteAlias* accetta un parametro, il nome dell'alias da cancellare e rende inaccessibile l'alias alla sessione.



DeleteAlias non rimuove un alias dal file di configurazione BDE se l'alias è stato scritto sul file da una chiamata precedente a *SaveConfigFile.* Per togliere l'alias dal file di configurazione dopo avere chiamato *DeleteAlias*, bisogna chiamare nuovamente *SaveConfigFile*.

I componenti session forniscono cinque metodi per recuperare le informazioni relative a un alias BDE, comprese le informazioni sul parametro e quelle sul driver. Essi sono:

- *GetAliasNames*, per elencare gli alias ai quali una sessione ha accesso.
- *GetAliasParams*, per elencare i parametri di un alias specificato.
- *GetAliasDriverName*, per restituire il nome del driver BDE utilizzato dall'alias.
- *GetDriverNames*, per restituire una lista di tutti i driver BDE disponibili per la sessione.
- *GetDriverParams*, per restituire parametri di driver relativi a un driver specificato.

Per ulteriori informazioni sull'utilizzo dei metodi per recuperare le informazioni di una sessione, consultare "Reperimento di informazioni su una sessione" qui di seguito. Per ulteriori informazioni sugli alias BDE e sui driver SQL Links con cui essi funzionano, consultare la Guida BDE in linea, BDE32.HLP.

Reperimento di informazioni su una sessione

Per recuperare le informazioni su una sessione e sui suoi componenti database si utilizzano i metodi informativi di una sessione. Ad esempio, un metodo recupera i nomi di tutti gli alias conosciuti dalla sessione, e un altro recupera i nomi delle tabelle associate a uno specifico componente database utilizzato dalla sessione. La Tabella 24.4 riassume i metodi informativi di un componente session:

Tabella 24.4 Metodi informativi correlati al database dei componenti session

Metodo	Scopo
GetAliasDriverName	Recupera il driver BDE dell'alias specificato di un database.
GetAliasNames	Recupera la lista degli alias BDE di un database.
GetAliasParams	Recupera la lista di parametri dell'alias BDE specificato di un database.
GetConfigParams	Recupera informazioni di configurazione dal file di configurazione BDE.
GetDatabaseNames	Recupera la lista degli alias BDE e i nomi di tutti i componenti *TDatabase* attualmente in uso.
GetDriverNames	Recupera i nomi di tutti i driver BDE installati.
GetDriverParams	Recupera la lista di parametri di un driver BDE specificato.
GetStoredProcNames	Recupera i nomi di tutte le procedure registrate relative a un database specificato.

Tabella 24.4 Metodi informativi correlati al database dei componenti session (continua)

Metodo	Scopo
GetTableNames	Recupera i nomi di tutte le tabelle di un database specificato corrispondenti a una struttura specificata.
GetFieldNames	Recupera i nomi di tutti i campi di una tabella specificata in un database specificato.

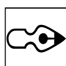
Salvo *GetAliasDriverName*, questi metodi restituiscono un set di valori in una lista di stringhe dichiarata e gestita dall'applicazione. (*GetAliasDriverName* restituisce una sola stringa, il nome del driver BDE attivo di un particolare componente database utilizzato dalla sessione).

Ad esempio, il seguente codice ricerca i nomi di tutti i componenti database e gli alias conosciuti alla sessione predefinita:

```
TStringList *List = new TStringList();
try
{
  Session->GetDatabaseNames(List);
  ⋮
}
catch (...)
{
  delete List;
  throw;
}
delete List;
```

Creazione di sessioni supplementari

È possibile creare sessioni per integrare la sessione predefinita. In fase di progettazione, si possono situare le sessioni supplementari su un modulo dati (o su una scheda), impostarne le proprietà nell'Object Inspector, scrivere per loro dei gestori di evento, oltre a scrivere il codice per chiamare i loro metodi. Anche in fase di esecuzione si possono creare sessioni, impostarne le proprietà e chiamarne i metodi.



Creare sessioni supplementari è facoltativo, salvo il caso in cui l'applicazione esegua query simultanee in un database, oppure l'applicazione sia di tipo multithread.

In fase di esecuzione per attivare la creazione dinamica di un componente session, seguire questi passi:

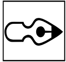
1. Dichiarare una variabile *TSession*.
2. Istanziare una nuova sessione usando l'operatore **new**.Questo operatore chiama il costruttore di *TSession* per creare e inizializzare una nuova sessione. Il costruttore assegna alla sessione una lista vuota di componenti database, imposta a **true** la proprietà *KeepConnections*, e inserisce la sessione nella lista gestita dal componente session list dell'applicazione.
3. Assegnare un nome univoco alla proprietà *SessionName* della nuova sessione a. Questa proprietà serve per per associare componenti database alla sessione. Per

ulteriori informazioni sulla proprietà *SessionName*, consultare "Attribuzione del nome a una sessione" a pagina 24-29.

4 Attivare la sessione e facoltativamente modificarne le proprietà .

È possibile anche creare e aprire sessioni tramite il metodo *OpenSession* di *TSessionList*. Usare *OpenSession* è più sicuro che utilizzare l'operatore **new**, perché *OpenSession* crea una sessione solo se questa non esiste già . Per informazioni su *OpenSession*, consultare "Gestione di sessioni multiple" a pagina 24-29.

Il seguente codice crea un nuovo componente session, gli assegna un nome e apre la sessione per svolgere delle operazioni su un database (qui non riportate). Dopo l'utilizzo, il componente viene distrutto mediante una chiamata al metodo *Free*.



Non cancellare mai la sessione predefinita.

```
TSession *SecondSession = new TSession(Form1);
try
{
  SecondSession->SessionName = "SecondSession";
  SecondSession->KeepConnections = false;
  SecondSession->Open();
  ⋮
}
__finally
{
  delete SecondSession;
};
```

Attribuzione del nome a una sessione

La proprietà *SessionName* di una sessione serve per dare il nome alla sessione, in modo che sia possibile associarle i database e i dataset. Nel caso della sessione predefinita, *SessionName* vale "Default", For ogni componente session aggiuntivo che si crea, è necessario dare un valore univoco alla sua proprietà *SessionName*.

I componenti database e dataset hanno proprietà *SessionName* che corrispondono alla proprietà *SessionName* di un componente session. Se si lascia vuota la proprietà *SessionName* di un componente database o dataset questa viene associata automaticamente alla sessione predefinita. È possibile anche assegnare a *SessionName* di un componente database o dataset un nome corrispondente al *SessionName* di un componente session che si crea.

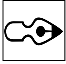
Il seguente codice utilizza il metodo *OpenSession* del componente predefinito *TSessionList*, *Sessions*, per aprire un nuovo componente session, assegnare al suo *SessionName* il nome "InterBaseSession", attivare la sessione e associare alla sessione *Database1*, un componente database esistente:

```
TSession *IBSession = Sessions->OpenSession("InterBaseSession");
Database1->SessionName = "InterBaseSession";
```

Gestione di sessioni multiple

Se si crea una singola applicazione che utilizza più thread per svolgere operazioni sul database, è necessario creare una sessione supplementare per ogni thread. La pagina

BDE sulla Component palette contiene un componente session che in fase di progettazione si può mettere in un modulo dati o su una scheda.



Quando si colloca un componente session, è necessario anche dare alla sua proprietà *SessionName* un valore univoco, per evitare conflitti con la proprietà *SessionName* della sessione predefinita.

Il fatto di inserire in fase di progetto un componente session presuppone che il numero di thread (e quindi di sessioni) richiesti dall'applicazione durante l'esecuzione sia stabile. Tuttavia, è più probabile che un'applicazione debba creare sessioni dinamicamente. Per creare delle sessioni in modo dinamico, bisogna chiamare in fase di esecuzione il metodo *OpenSession* dell'oggetto globale *Sessions*.

OpenSession richiede un singolo parametro, un nome di sessione univoco rispetto a tutti i nomi di sessione dell'applicazione. Con questo codice si crea dinamicamente e si attiva una nuova sessione con un nome generato univocamente:

```
Sessions->OpenSession("RunTimeSession" + IntToStr(Sessions->Count + 1));
```

Questa istruzione genera un nome univoco di una nuova sessione recuperando il numero di sessioni corrente e aggiungendo uno al valore trovato. Notare che se si creano e distruggono dinamicamente delle sessioni in fase di esecuzione, l'esempio qui riportato non si comporta come previsto. Tuttavia, l'esempio mostra come utilizzare le proprietà e i metodi di *Sessions* per gestire sessioni multiple.

Sessions è una variabile di tipo *TSessionList* che viene automaticamente istanziata nel caso di applicazioni database BDE. Le proprietà e i metodi di *Sessions* si utilizzano per tenere il conto di sessioni multiple nelle applicazioni database multithread. La Tabella 24.5 riassume le proprietà e i metodi del componente *TSessionList*:

Tabella 24.5 Proprietà e metodi di TSessionList

Proprietà o metodo	Scopo
Count	Restituisce il numero di sessioni, sia attive che non attive, presenti nella lista di sessioni.
FindSession	Cerca la sessione con il nome specificato e restituisce un puntatore ad essa, oppure restituisce NULL se la con il nome specificato non esiste. Se viene passato un nome di sessione vuoto, *FindSession* restituisce un puntatore alla sessione predefinita, *Session*.
GetSessionNames	Mette in una lista di stringhe i nomi di tutti i componenti session correnti. La procedura aggiunge sempre almeno una stringa, "Default", per la sessione predefinita.
List	Restituisce il componente session di un nome di sessione specificato. Se non c'è alcuna sessione con il nome specificato, viene sollevata un'eccezione.
OpenSession	Crea e attiva una nuova sessione oppure riattiva una sessione esistente relativamente a un nome di sessione specificato.
Sessions	Accede alla lista di sessioni in base al valore ordinale.

Come esempio dell'uso delle proprietà e dei metodi *Sessions* in un'applicazione multithread, si consideri che cosa accade quando si intende aprire una connessione con un database. Per stabilire se la connessione è già in corso, si ricorre alla proprietà *Sessions* per far passare ciascuna sessione presente nella lista sessioni a partire dalla

sessione predefinita. Per ciascun componente session, si esamina la proprietà *Databases* per vedere se il database in questione è aperto. Se si scopre che un altro thread sta già usando il database richiesto, si esamina la sessione successiva nella lista.

Se un thread esistente non utilizza il database, allora si può stabilire la connessione all'interno di quella sessione.

Se, d'altra parte, tutti i thread esistenti utilizzano il database, è necessario aprire una nuova sessione in cui aprire un'altro collegamento con il database.

Se in un'applicazione multithread si replica un modulo dati contenente una sessione, e ogni thread contiene una propria copia del modulo dati, con la proprietà *AutoSessionName* ci si può accertare che tutti i dataset del modulo dati utilizzino la sessione corretta. L'impostazione di *AutoSessionName* a **true** fa in modo che la sessione quando viene creata in fase di esecuzione generi dinamicamente il proprio nome univoco. Il nome poi viene assegnato a ciascun dataset del modulo dati, ridefinendo qualunque nome di sessione esplicitamente impostato. In questo modo si ha la certezza che ciascun thread abbia la propria sessione e che ciascun dataset utilizzi la sessione nel proprio modulo dati.

Utilizzo delle transazioni con BDE

Per impostazione predefinita, BDE fornisce un controllo di transazione implicito alle applicazioni. Quando un'applicazione è soggetta a controllo di transazione implicito, per ciascun record di un dataset che viene scritto nel database sottostante si ricorre a una transazione separata. Le transazioni implicite garantiscono il minimo di conflitti di aggiornamento di record sia una vista coerente del database. D'altra parte, siccome per ogni riga di dati scritti in un database si ricorre a una transazionedistinta, il controllo di transazione implicito può portare a un eccessivo traffico di rete e a un rallentamento delle prestazioni. Inoltre Il controllo di transazione implicito non protegge le operazioni logiche che impegnano più di un record.

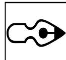
Se si controllano esplicitamente le transazioni, si può scegliere il momento migliore per per farle partire, impegnarle e per fare il rollback. Nello sviluppo di applicazioni in ambiente multiutente, in particolare quando le applicazioni vengono eseguite su un server SQL remoto, si dovrebbero controllare le transazioni esplicitamente.

Nelle applicazioni database basate su BDE ci sono due modi che si escludono reciprocamente per il controllo esplicito delle transazioni

- Il controllo viene fatto mediante il componente database. Il vantaggio principale dell'impiego dei metodi e delle proprietà di un componente database è quello di fornire un'applicazione pulita e portabile, indiprendente dal database o dal server. Questo tipo di controllo di transazione è supportato da tutti i componenti di connessione al database ed è descritto in "Gestione delle transazioni" a pagina 21-6.
- Utilizzare una passthrough SQL in un componente query per passare istruzioni SQL direttamente a un server SQL oppure ODBC remoto. Il vantaggio principale del passthrough SQL sta nella possibilità di sfruttare le funzionalità avanzate di

gestione di transazioni di un particolare server di database, come per esempio la cache di schema. Per capire i vantaggi del modello di gestione delle transazioni del server di cui si dispone, consultare la documentazione relativa. Per ulteriori informazioni sull'utilizzo di passthrough SQL, consultare "Utilizzo di passthrough SQL" qui di seguito.

Quando si lavora su database locali, il componente database serve solo per creare transazioni esplicite (i database locali non supportano passthrough SQL). Tuttavia, l'impiego di transazioni locali è soggetto a certe limitazioni. Per ulteriori informazioni sull'utilizzo di transazioni locali, consultare "Utilizzo delle transazioni locali" a pagina 24-33.



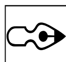
È possibile rendere minimo il numero di transazioni necessarie registrando gli aggiornamenti nella cache. Per ulteriori informazioni sugli aggiornamenti registrati nella cache, consultare "Utilizzo di un dataset client per registrare in cache gli aggiornamenti" a pagina 27-17 e "Utilizzo di BDE per gli aggiornamenti in cache" a pagina 24-33.

Utilizzo di passthrough SQL

Con passthrough SQL, si utilizza un componente *TQuery*, *TStoredProc* o *TUpdateSQL* per inviare direttamente a un server di database remoto un'istruzione di controllo di transazione SQL. BDE non elabora l'istruzione SQL. Usando passthrough SQL si possono sfruttare direttamente i controlli di transazione offerti dal server, specialmente quando sono di tipo non-standard.

Per per controllare una transazione mediante passthrough SQL, è necessario

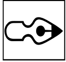
- Installare i driver SQL Links corretti. Se installando C++Builder si era scelta l'opzione "Typical", tutti i driver SQL Links risultano già installati correttamente.
- Configurare il protocollo di rete. Consultare l'amministratore di rete per ulteriori informazioni.
- Avere accesso a un database situato su un server remoto.
- Mediante SQL Explorer impostare SQLPASSTHRU MODE a NOT SHARED. SQLPASSTHRU MODE specifica se le istruzioni BDE e passthrough SQL possono condividere gli stessi collegamenti al database. Nella maggior parte dei casi, l'impostazione di SQLPASSTHRU MODE è SHARED AUTOCOMMIT. Tuttavia, è impossibile l'uso condiviso di connessioni con il database quando si utilizzano istruzioni di controllo di transazione. Per ulteriori informazioni su modi SQLPASSTHRU, consultare il file di guida del programma di utilità BDE Administration.



Quando SQLPASSTHRU MODE è NOT SHARED, è necessario utilizzare dei componenti database distinti per i dataset che passano istruzioni di transazione SQL al server e per quelli che non lo fanno.

Utilizzo delle transazioni locali

BDE supporta transazioni locali con tabelle Paradox, dBASE, Access e FoxPro. Dal punto di vista di chi programma, non c'è differenza fra una transazione locale e una transazione con un server di database remoto.



Quando si utilizzano transazioni con tabelle locali Paradox, dBASE, Access e FoxPro, impostare *TransIsolation* a *tiDirtyRead* invece di utilizzare il valore predefinito di *tiReadCommitted*. Se nel caso di tabelle locali l'impostazione di *TransIsolation* è comunque diversa da *tiDirtyRead* viene restituito un errore BDE.

Quando si inizia una transazione con una tabella locale, gli aggiornamenti eseguiti vengono registrati. Ogni record registrato contiene il buffer di un record. Quando una transazione è attiva, i record da aggiornare sono bloccati fino a quando la transazione non è conclusa o annullata. In caso di rollback, i vecchi buffer dei record vengono applicati ai record aggiornati in modo da riportarli allo stato precedente.

Le transazioni locali sono più limitate delle transazioni con i server SQL o i driver ODBC. In particolare, per le transazioni locali valgono le seguenti limitazioni:

- Non è fornito il crash recovery automatico.
- Non sono supportate le istruzioni di definizione dati.
- Non si possono effettuare transazioni con tabelle temporanee.
- Il livello di *TransIsolation* può essere solo *tiDirtyRead*.
- Per Paradox, le transazioni locali possono essere eseguite solo su tabelle che abbiano indici validi. Non è possibile il rollback dei dati con tabelle Paradox prive di indici.
- Si può bloccare e modificare solo un numero limitato di record. Con tabelle Paradox, il limite è 255 record. Con tabelle dBASE il limite è 100.
- Le non si possono eseguire transazioni con il driver ASCII BDE.
- Se si chiude un cursore su una tabella durante una transazione se ne determina il rollback a meno che:
  - Non ci siano più tabelle aperte.
  - Il cursore venga chiuso su una tabella che non ha subito modifiche.

Utilizzo di BDE per gli aggiornamenti in cache

L'approccio consigliato per utilizzare gli aggiornamenti in cache consiste nell'usare un dataset client (*TBDEClientDataSet*) o nel collegare il dataset BDE a un dataset client utilizzando un provider di dataset. I vantaggi dell'utilizzo di un dataset client sono discussi in "Utilizzo di un dataset client per registrare in cache gli aggiornamenti" a pagina 27-17.

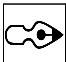
Nei casi semplici, tuttavia, si può invece decidere di registrare gli aggiornamenti in cache ricorrendo a BDE. I dataset per BDE e i componenti *TDatabase* dispongono di proprietà , di metodi e di eventi per gestire gli aggiornamenti in cache. Nella maggior

parte dei casi, questi corrispondono direttamente alle proprietà , ai metodi e agli eventi che si usano con i dataset client e con i dataset provider quando si ricorre a un dataset client per la gestione degli aggiornamenti in cache. La seguente tabella elenca queste proprietà , questi eventi e questi metodi accanto alle proprietà , i metodi e gli eventi corrispondenti di *TBDEClientDataSet*:

Tabella 24.6 Proprietà , metodi ed eventi degli aggiornamenti in cache

Su dataset BDE (o TDatabase)	Su TBDEClientDataSet	Scopo
CachedUpdates	Non necessario per i dataset client, che usano sempre gli aggiornamenti in cache.	Determina se gli aggiornamenti in cache sono attivi per il dataset.
UpdateObject	Usare un gestore di evento *BeforeUpdateRecord*, o, se si utilizza *TClientDataSet*, usare la proprietà *UpdateObject* del dataset sorgente per BDE.	Specifica l'oggetto update per l'aggiornamento di dataset di sola lettura.
UpdatesPending	*ChangeCount*	Indica se la cache locale contiene record aggiornati che devono essere applicati al database.
UpdateRecordTypes	*StatusFilter*	Indica il genere di record aggiornati da rendere visibili quando si applicano gli aggiornamenti messi nella cache.
UpdateStatus	*UpdateStatus*	Indica se un record è invariato, modificato, inserito o cancellato.
OnUpdateError	*OnReconcileError*	Evento per la gestione record per record degli errori di aggiornamento -
OnUpdateRecord	*BeforeUpdateRecord*	Evento per l'elaborazione record per record degli aggiornamenti.
ApplyUpdates *ApplyUpdates* (database)	*ApplyUpdates*	Applica al database i record presenti nella cache locale.
CancelUpdates	*CancelUpdates*	Elimina dalla cache locale tutti gli aggiornamenti senza applicarli.
CommitUpdates	*Reconcile*	Svuota la cache di aggiornamento dopo la conclusione favorevole degli aggiornamenti.
FetchAll	*GetNextPacket* (e *PacketRecords*)	Copia i record nella cache locale per modificarli e aggiornarli.
RevertRecord	*RevertRecord*	Annulla gli aggiornamenti del record attivo se non sono stati ancora applicati.

Per una panoramica dei processi di aggiornamento mediante la cache, consultare "Panoramica sull'utilizzo degli aggiornamenti in cache" a pagina 27-18.



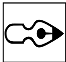
Anche se si utilizza un dataset client per la gestione degli aggiornamenti in cache, è utile leggere la sezione sugli oggetti update a pagina 24-41. Si possono utilizzare gli oggetti update nel gestore di evento *BeforeUpdateRecord* di *TBDEClientDataSet* o di

TDataSetProvider per applicare aggiornamenti provenienti da procedure registrate o da query su più tabelle.

Attivazione degli aggiornamenti in cache con BDE

Per utilizzare BDE per gli aggiornamenti in cache, il dataset per BDE deve indicare l'intenzione di mettere gli aggiornamenti nella cache. Lo si specifica impostando la proprietà *CachedUpdates* a **true**. Quando l'aggiornamento dei dati memorizzati nella cache è attivo, nella memoria locale viene posta una copia di tutti i record. Gli utenti visualizzano e modificano la copia locale dei dati. Nella cache vengono registrate anche le modifiche, gli inserimenti e le cancellazioni che si accumulano in memoria fino a quando l'applicazione non li applica al server di database. Se le modifiche vanno a buon fine, nella cache i record modificati vengono scaricati dalla memoria.

Il dataset mette nella cache tutti gli aggiornamenti, fino a quando non si imposta *CachedUpdates* a **false**. L'applicazione delle modifiche presenti nella cache non disabilita gli ulteriori aggiornamenti memorizzati, ma trascrive solo nel database l'insieme di modifiche corrente e le scarica dalla memoria. Se si annullano gli aggiornamenti chiamando *CancelUpdates*, tutte le modifiche presenti nella cache vengono annullate, senza però impedire al dataset di mettere nella cache le modifiche successive.



Se si disattiva l'aggiornamento in cache impostando *CachedUpdates* a **false**, le eventuali modifiche non ancora applicate vengono eliminate senza notifica. Per evitare la perdita di modifiche, testare la proprietà *UpdatesPending* prima di disattivare l'aggiornamento in cache.

Applicazione degli aggiornamenti in cache basati su BDE

L'applicazione degli aggiornamenti è un processo in due fasi che dovrebbe verificarsi nel contesto della transazione di un componente database, in modo che l'applicazione possa ristabilirsi convenientemente da errori. Per informazioni sulla gestione delle transazioni nel caso dei componenti database, consultare "Gestione delle transazioni" a pagina 21-6.

Applicando gli aggiornamenti sotto il controllo delle transazioni si verifica quanto segue:

1. Viene attivata una transazione con il database.
2. Gli aggiornamenti nella cache vengono trascritti nel database (fase 1). Se si fornisce un evento *OnUpdateRecord*, questo viene attivato una volta ogni volta che nel database viene trascritto un record. Se la trascrizione produce un errore, viene attivato l'evento *OnUpdateError*, se previsto.
3. La transazione viene confermata se la trascrizione va a buon fine, o annullata in caso contrario:

   Se la trascrizione va a buon fine:

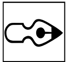
   - Le modifiche apportate al database vengono confermate e la transazione si conclude.

- Gli aggiornamenti nella cache vengono confermati e il buffer interno della cache viene scaricato (fase 2).

Se la trascrizione non va a buon fine:

- Le modifiche al database vengono annullate e la transazione si conclude.
- Gli aggiornamenti presenti nella cache non vengono confermati e rimangono nella cache interna.

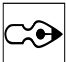
Per informazioni sulla creazione e sull'impiego dei gestori di evento *OnUpdateRecord*, consultare "Creazione di un gestore di evento OnUpdateRecord" a pagina 24-38. Per informazioni sulla gestione degli errori di aggiornamento che si verificano quando si applicano gli aggiornamenti in cache, consultare "Gestione degli errori di aggiornamento" a pagina 24-39.



L'applicazione degli aggiornamenti in cache è un'operazione particolarmente delicata quando si ha a che fare con più dataset collegati secondo una relazione master/detail perché è significativo l'ordine di applicazione degli aggiornamenti a ciascun dataset. Di solito, è necessario aggiornare le tabelle master prima di quelle di dettaglio, salvo quando si ha a che fare con record cancellati, perché allora l'ordine va invertito. A causa di questa difficoltà , si raccomanda vivamente l'impiego di dataset client quando si mettono in cache gli aggiornamenti di una scheda master/detail. I dataset client gestiscono automaticamente tutte le questioni di ordinamento nelle relazioni master/detail.

Ci sono due modi per applicare gli aggiornamenti basati su BDE:

- Un modo è quello di applicare gli aggiornamenti utilizzando un componente database e chiamando il suo metodo *ApplyUpdates*. Questo è l'approccio più semplice, perché il database gestisce tutti i dettagli del processo di transazione e scarica la cache ad aggiornamento concluso.
- È possibile aggiornare un singolo dataset chiamando i suoi metodi *ApplyUpdates* e *CommitUpdates*. In questo caso, è necessario codificare esplicitamente la transazione che interessa il processo di aggiornamento, come pure chiamare esplicitamente *CommitUpdates* per confermare gli aggiornamenti provenienti dalla cache.



Per applicare gli aggiornamenti da una procedura registrata o da una query SQL che non restituisce un set risultato "live", è necessario utilizzare *TUpdateSQL* per specificare come eseguire gli aggiornamenti. Per aggiornamenti a concatenamenti (query implicanti due o più tabelle), è necessario fornire un oggetto *TUpdateSQL* per ogni tabella coinvolta, e utilizzare il gestore di evento *OnUpdateRecord* per richiamare questi oggetti in modo da eseguire gli aggiornamenti. Consultare "Aggiornamento di dataset mediante oggetti update" a pagina 24-41 per i dettagli.

Applicazione degli aggiornamenti nella cache mediante un database

Per applicare nel contesto di una connessione con un database gli aggiornamenti presenti nella cache a uno o più dataset, si usa il metodo *ApplyUpdates* del componente database. Il seguente codice applica degli aggiornamenti al dataset *CustomersQuery* in risposta all'evento clic di un pulsante:

```
void __fastcall TForm1::ApplyButtonClick(TObject *Sender)
```

```
{
  // for local databases such as Paradox, dBASE, and FoxPro
  // set TransIsolation to DirtyRead
  if (!Database1->IsSQLBased && Database1->TransIsolation != tiDirtyRead)
    Database1->TransIsolation = tiDirtyRead;
  Database1->ApplyUpdates(&CustomersQuery,0);
}
```

Con le righe precedenti si trascrivono nel database gli aggiornamenti presenti nella cache nel contesto di una transazione generata automaticamente. Se l'operazione ha successo, la transazione viene confermata e poi vengono confermati gli aggiornamenti nella cache. In caso contrario, si ha l'annullamento della transazione e la cache rimane invariata. In questo secondo caso, gli errori dovuti agli aggiornamenti in cache andrebbero gestiti mediante l'evento *OnUpdateError* del dataset. Per ulteriori informazioni sulla gestione degli errori di aggiornamento, consultare "Gestione degli errori di aggiornamento" a pagina 24-39.

Il vantaggio principale della chiamata del metodo *ApplyUpdates* di un componente database è la possibilità di aggiornamento di qualsiasi numero di componenti dataset associati al database. I due argomenti per il metodo *ApplyUpdates* di un database sono un array di *TDBDataSet*, e l'indice dell'ultimo dataset nell'array. Per applicare gli aggiornamenti per più di un dataset, creare un array locale di puntatori ai dataset. Ad esempio, il seguente codice applica gli aggiornamenti di due query:

```
TDBDataSet* ds[] = {CustomerQuery, OrdersQuery};
if (!Database1->IsSQLBased && Database1->TransIsolation != tiDirtyRead)
  Database1->TransIsolation = tiDirtyRead;
Database1->ApplyUpdates(ds,1);
```

Applicazione degli aggiornamenti in cache con i metodi dei componenti dataset

È possibile aggiornare singoli dataset per BDE tramite i metodi *ApplyUpdates* e *CommitUpdates* del dataset. Ognuno di questi metodi racchiude una fase del processo di aggiornamento:

1 *ApplyUpdates* trascrive in un database le modifiche presenti nella cache (fase 1).
2 *CommitUpdates* scarica (cancella) la cache interna quando la trascrizione va a buon fine (fase 2).

Il seguente codice illustra come si applicano gli aggiornamenti all'interno di una transazione riguardante il dataset *CustomerQuery*:

```
void __fastcall TForm1::ApplyButtonClick(TObject *Sender)
{
  Database1->StartTransaction();
  try
  {
    if (!Database1->IsSQLBased && Database1->TransIsolation != tiDirtyRead)
      Database1->TransIsolation = tiDirtyRead;
    CustomerQuery->ApplyUpdates(); // try to write the updates to the database
    Database1->Commit(); // on success, commit the changes
  }
  catch (...)
```

```
    {
      Database1->Rollback(); // on failure, undo any changes
      throw; // throw the exception again to prevent a call to CommitUpdates
    }
    CustomerQuery->CommitUpdates(); // on success, clear the internal cache
  }
```

Se durante la chiamata di *ApplyUpdates* viene sollevata un'eccezione, si ha l'annullamento della transazione. Il rollback assicura che nella tabella di database non vengano trascritte modifiche. L'istruzione *throw* nel blocco **try**... **catch** *solleva nuovamente* l'eccezione, evitando la chiamata a *CommitUpdates*. Siccome *CommitUpdates* non viene chiamato, la cache interna degli aggiornamenti non viene cancellata consentendo di gestire la condizione di errore e, se è il caso, di ritentare l'aggiornamento.

Creazione di un gestore di evento OnUpdateRecord

Per applicare gli aggiornamenti memorizzati nella sua cache, un dataset per BDE esamina in sequenza le modifiche presenti nella cache, e cerca di trascriverle nei record corrispondenti nella tabella sottostante. Mentre l'aggiornamento per ogni record modificato, cancellato o inserito sta per essere trascritto, viene attivato l'evento *OnUpdateRecord* del componente dataset.

Se si prepara un gestore per l'evento *OnUpdateRecord* si ha la possibilità di eseguire determinate azioni prima dell'effettiva trascrizione dell'aggiornamento del record corrente. Tra queste vi possono essere una convalida speciale dei dati, l'aggiornamento di altre tabelle, la sostituzione speciale di parametri, o l'esecuzione di più oggetti update. Un gestore di evento *OnUpdateRecord* consente di controllare meglio il processo di aggiornamento.

Ecco lo schema del codice di un gestore di evento *OnUpdateRecord*:

```
void __fastcall TForm1::DataSetUpdateRecord(TDataSet *DataSet,
  TUpdateKind UpdateKind, TUpdateAction &UpdateAction)
{
  // Perform updates here...
}
```

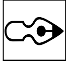
Il parametro *DataSet* specifica il dataset memorizzato nella cache con gli aggiornamenti.

Il parametro *UpdateKind* indica il tipo di aggiornamento da eseguire sul record attivo. I valori di *UpdateKind* sono *ukModify*, *ukInsert* e *ukDelete*. Se si utilizza un oggetto update, è necessario passare questo parametro all'oggetto update quando l'aggiornamento viene trascritto. Può anche essere necessario esaminare il parametro se il gestore esegue qualche elaborazione speciale in base al genere di aggiornamento.

Il parametro *UpdateAction* indica se l'aggiornamento è stato applicato. I valori di *UpdateAction* sono *uaFail* (l'impostazione predefinita), *uaAbort*, *uaSkip*, *uaRetry*, *uaApplied*. Se il gestore di evento porta a buon fine l'aggiornamento, prima di uscire il parametro deve diventare *uaApplied*. Se si decide di non aggiornare il record attivo, si deve il valore di *uaSkip* per lasciare nella cache le modifiche non attuate. Se non si modifica il valore di *UpdateAction*, l'intera operazione di aggiornamento del dataset

si interrompe e viene sollevata un'eccezione. È possibile sopprimere il messaggio di errore (sollevando un'eccezione tacita) se si cambia *UpdateAction* in *uaAbort*.

Oltre a questi parametri, normalmente si utilizzano le proprietà *OldValue* e *NewValue* del componente campo associato al record attivo. *OldValue* dà il valore originario del campo che è stato prelevato dal database. Può essere utile nell'individuare il record di database da aggiornare. *NewValue* è il valore modificato dell'aggiornamento che si cerca di trascrivere.



Il gestore di evento *OnUpdateRecord*, proprio come il gestore di evento *OnUpdateError* o *OnCalcFields*, non dovrebbe mai chiamare metodi che modificano il record attivo di un dataset.

Il seguente esempio illustra come utilizzare questi parametri e queste proprietà . Per applicare gli aggiornamenti utilizza un componente *TTable* chiamato *UpdateTable*. Nella pratica, è più facile utilizzare un oggetto update, ma usare una tabella illustra le possibilità più chiaramente.

```
void __fastcall TForm1::EmpAuditUpdateRecord(TDataSet *DataSet,
  TUpdateKind UpdateKind, TUpdateAction &UpdateAction)
{
  if (UpdateKind == ukInsert)
  {
    TVarRec values[2];
    for (int i = 0; i < 2; i++)
      values[i] = DataSet->Fields->Fields[i]->NewValue;
    UpdateTable->AppendRecord(values, 1);
  }
  else
  {
    TLocateOptions lo;
    lo.Clear();
    if (UpdateTable->Locate("KeyField", DataSet->Fields->Fields[0]->OldValue, lo))
      switch (UpdateKind)
      {
        case ukModify:
          UpdateTable->Edit();
          UpdateTable->Fields->Fields[1]->Value = DataSet->Fields->Fields[1]->Value;
          UpdateTable->Post();
          break;
        case ukDelete:
          UpdateTable->Delete();
          break;
```

Gestione degli errori di aggiornamento

Borland Database Engine (BDE) controlla specificatamente i conflitti di aggiornamento e altre condizioni quando tenta di applicare gli aggiornamenti e riporta gli errori. L'evento *OnUpdateError* del componente dataset consente di rilevare e di rispondere agli errori. Se si utilizzano gli aggiornamenti in cache si dovrebbe creare un gestore di questo evento. In caso contrario, quando si verifica un errore, l'intera operazione di aggiornamento non va a buon fine.

Ecco lo schema di codice di un gestore di evento *OnUpdateError*:

```
void __fastcall TForm1::DataSetUpdateError(TDataSet *DataSet,
  EDatabaseError *E, TUpdateKind UpdateKind, TUpdateAction &UpdateAction)
{
  // Respond to errors here...
}
```

DataSet fa riferimento al dataset a cui sono applicati gli aggiornamenti. È possibile utilizzarlo in fase di gestione degli errori per accedere a nuovi e a vecchi valori. I valori originali dei campi di ciascun record sono memorizzati nella proprietà di sola lettura *TField* chiamata *OldValue*, mentre quelli modificati sono memorizzati nella proprietà analoga di *TField* chiamata *NewValue*. Questi valori sono l'unico modo per esaminare e modificare i valori di aggiornamento del gestore di evento



Non chiamare metodi di dataset (come *Next* e *Prior*) che modificano il record attivo, perché si farebbe entrare in un loop infinito il gestore di evento.

Di solito il parametro *E* è del tipo *EDBEngineError*. Da questo tipo di eccezione, si può estrarre un messaggio di errore che è possibile mostrare agli utenti nel gestore di errore. Ad esempio, si potrebbe usare questo codice per visualizzare un messaggio di errore nell'intestazione di una finestra di dialogo:

```
ErrorLabel->Caption = E->Message;
```

Questo parametro è anche utile per stabilire la causa effettiva dell'errore di aggiornamento. È possibile estrarre degli specifici codici di errore da *EDBEngineError* e in base ad essi eseguire l'azione appropriata.

Il parametro *UpdateKind* descrive il tipo di aggiornamento che ha generato l'errore. A meno che il gestore di errore non esegua azioni speciali in base a il tipo di aggiornamento eseguito, probabilmente il codice non farà uso di questo parametro.

La seguente tabella elenca i possibili valori di *UpdateKind*:

Tabella 24.7 Valori di UpdateKind

Valore	Significato
ukModify	La modifica di un record esistente ha causato un errore.
ukInsert	L'inserimento di un nuovo record ha causato un errore.
ukDelete	La cancellazione di un record esistente ha causato un errore.

UpdateAction indica a BDE come portare avanti il processo di aggiornamento quando il gestore di evento si chiude. Quando il gestore di evento viene chiamato, il valore di questo parametro è sempre *uaFail*. In base alla condizione di errore del record che ha causato l'errore e all'azione intrapresa per rimediare, solitamente prima di uscire dal gestore a *UpdateAction* viene attribuito un valore diverso:

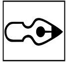
- Se il gestore di errore può correggere la condizione di errore che ha fatto richiamare il gestore, si imposta *UpdateAction* in modo che svolga l'azione appropriata all'uscita. Per le condizioni di errore che è possibile correggere, impostare *UpdateAction* a *uaRetry* per richiedere nuovamente l'aggiornamento del record..

- Quando l'impostazione è *uaSkip*, l'aggiornamento della riga che ha causato l'errore viene saltato e l'aggiornamento del record rimane nella cache dopo il completamento di tutti gli altri aggiornamenti.
- Sia *uaFail* che *uaAbort* determinano la conclusione dell'intera operazione di aggiornamento. *uaFail* solleva un'eccezione e visualizza un messaggio di errore. *uaAbort* solleva un'eccezione silenziosa (non visualizza un messaggio di errore).

Il seguente codice visualizza un gestore di evento *OnUpdateError* che verifica se l'errore di aggiornamento è relativo a una violazione di chiave e, in caso positivo, imposta il parametro *UpdateAction* a *uaSkip*:

```
// include BDE.hpp in your unit file for this example

void __fastcall TForm1::DataSetUpdateError(TDataSet *DataSet,
  EDatabaseError *E, TUpdateKind UpdateKind, TUpdateAction &UpdateAction)
{
  UpdateAction = uaFail // initialize to fail the update
  if (E->ClassNameIs("EDBEngineError"))
  {
    EDBEngineError *pDBE = (EDBEngineError *)E;
    if (pDBE->Errors[pDBE->ErrorCount - 1]->ErrorCode == DBIERR_KEYVIOL)
      UpdateAction = uaSkip; // Key violation, just skip this record
  }
}
```



Se durante l'applicazione degli aggiornamenti in cache si verifica un errore, viene *sollevata* un'eccezione e appare un messaggio di errore. Salvo il caso in cui *ApplyUpdates* viene chiamato dall'interno di un ciclo **try...catch**, il messaggio di errore visualizzato all'utente da dentro il gestore di evento *OnUpdateError* può apparire ancora una volta. Per evitare la ripetizione bisogna dare a *UpdateAction* il valore *uaAbort* e disattivare la comparsa del messaggio di errore di sistema.

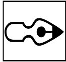
Aggiornamento di dataset mediante oggetti update

Quando il dataset per BDE rappresenta una procedura registrata o una query che non restituisce risultati "live", non si possono effettuare aggiornamenti direttamente dal dataset. Infatti si potrebbero verificare problemi quando si utilizza un dataset client per mettere in cache gli aggiornamenti. In ambedue i casi, si evita il problema utilizzando un oggetto update:

1. Se si utilizza un dataset client, ricorrere a un componente provider esterno con *TClientDataSet*, piuttosto che a *TBDEClientDataSet*. Questo per consentire di impostare la proprietà *UpdateObject* del dataset sorgente per BDE (punto 3).
2. Aggiungere un componente *TUpdateSQL* al modulo dati del dataset per BDE.
3. Impostare la proprietà *UpdateObject* del componente dataset per BDE al componente *TUpdateSQL* nel modulo dati.
4. Specificare le istruzioni SQL necessarie per eseguire gli aggiornamenti tramite le proprietà *ModifySQL*, *InsertSQL* e *DeleteSQL* dell'oggetto update. L'editor Update SQL aiuta a scrivere queste istruzioni.
5. Chiudere il dataset.

6 Assegnare il valore **true** alla proprietà *CachedUpdates* del componente dataset o collegare il dataset con il dataset client utilizzando un provider di dataset.

7 Riaprire il dataset.



Talvolta, è necessario utilizzare più oggetti update. Ad esempio, quando si aggiorna una join su più tabelle o una procedura registrata che rappresenta dati provenienti da più dataset, bisogna fornire un oggetto *TUpdateSQL* a ogni tabella da aggiornare. Quando si utilizzano più oggetti update, non si può semplicemente associare l'oggetto update al dataset tramite la proprietà *UpdateObject*. Invece, si deve chiamare manualmente l'oggetto update da un gestore di evento *OnUpdateRecord* (se si utilizza BDE per memorizzare gli aggiornamenti nella cache) o da un gestore di evento *BeforeUpdateRecord* (se si utilizza un dataset client).

L'oggetto update in effetti incorpora tre componenti *TQuery*, ognuno dei quali esegue una sola operazione di aggiornamento. Uno fornisce un'istruzione SQL UPDATE per la modifica di record esistenti; un secondo fornisce un'istruzione INSERT per aggiungere nuovi record a una tabella; e un terzo fornisce un'istruzione DELETE per eliminare i record da una tabella.

Quando si mette un componente update in un modulo dati, i componenti query che esso racchiude non si vedono. Essi vengono creati in fase di esecuzione dal componente update in base a tre proprietà di aggiornamento per le quali vanno fornite le istruzioni SQL:

- *ModifySQL* specifica l'istruzione UPDATE.
- *InsertSQL* specifica l'istruzione INSERT.
- *DeleteSQL* specifica l'istruzione DELETE.

In fase di esecuzione, quando il componente update viene usato per trascrivere gli aggiornamenti:

1 Seleziona un'istruzione SQL da eseguire a seconda che il record attivo venga modificato, inserito o cancellato.

2 Fornisce valori del parametro all'istruzione SQL.

3 Prepara ed esegue l'istruzione SQL in modo da farle portare a termine l'aggiornamento specificato.

Creazione di istruzioni SQL per componenti update

Per aggiornare un record di un dataset associato, un oggetto update utilizza una di tre istruzioni SQL. Ogni oggetto update può aggiornare solo una singola tabella, cosicché ciascuna istruzione di aggiornamento dell'oggetto deve fare riferimento alla stessa tabella base.

Le tre istruzioni SQL rispettivamente cancellano, inseriscono e modificano i record nella cache. Le istruzioni vanno fornite sotto forma di proprietà *DeleteSQL*, *InsertSQL* e *ModifySQL*. I valori possono essere forniti in fase di progetto o di esecuzione. Ad esempio, il seguente codice specifica in fase di esecuzione il valore della proprietà *DeleteSQL*:

```
UpdateSQL->DeleteSQL->Clear();
UpdateSQL->DeleteSQL->Add(”DELETE FROM Inventory I”);
```

```
UpdateSQL->DeleteSQL->Add("WHERE (I.ItemNo = :OLD_ItemNo)");
```

In fase di progettazione, l'editor Update SQL aiuta a scrivere le istruzioni SQL con cui si applicano gli aggiornamenti.

Gli oggetti update forniscono una connessione di parametro automatica nel caso dei parametri che referenziano i valori di campo originali e aggiornati del dataset. Di solito, quindi, quando si scrivono le istruzioni SQL si inseriscono parametri con nomi che hanno un formato speciale. Per informazioni sull'uso di questi parametri, consultare "Sostituzione dei parametri nelle istruzioni di aggiornamento SQL" a pagina 24-44.

Utilizzo di Update SQL editor

Per scrivere le istruzioni SQL di un componente update,

1 Mediante l'Object Inspector, selezionare il nome dell'oggetto update nella lista a discesa della proprietà *UpdateObject* del dataset. In questo modo, l'Update SQL editor che verrà richiamato nel passo successivo è in grado di determinare i valori predefiniti da utilizzare con le opzioni di generazione SQL.
2 Fare clic con il pulsante destro sull'oggetto update e selezionare l'UpdateSQL Editor nel menu di scelta rapida. Appare l'editor Update SQL, con cui si creano le istruzioni SQL delle proprietà *ModifySQL*, *InsertSQL* e *DeleteSQL* dell'oggetto update in base al dataset inerente e ai valori che gli vengono forniti.

L'editor Update SQL ha due pagine. La pagina Options appare quando l'editor viene richiamato. Per selezionare la tabella da aggiornare utilizzare la casella combinata Table Name. Quando si indica il nome della tabella, nelle caselle di riepilogo Key Fields e Update Fields appaiono tutte le colonne disponibili.

La casella di riepilogo Update Fields indica quali colonne aggiornare. Quando si specifica la tabella, appaiono selezionate tutte le colonne della casella di riepilogo Update Fields. Si possono selezionare più campi in base alle esigenze.

La casella di riepilogo Key Fields serve per specificare le colonne da usare come chiave nell'aggiornamento. Nel caso di tabelle Paradox, dBASE e FoxPro, le colonne specificate devono corrispondere a un indice esistente, cosa non obbligatoria nel caso di database SQL remoti. Invece di impostare Key Fields, facendo clic sul pulsante Primary Keys si scelgono i campi chiave di aggiornamento in base all'indice principale della tabella. Facendo clic su Dataset Defaults si riportano le liste di selezione allo stato originario: tutti i campi sono selezionati come chiavi e tutti i campi sono selezionati per l'aggiornamento.

Spuntare la casella di controllo Quote Field Names se il server vuole che i nomi di campo siano messi tra virgolette.

Una volta specificata la tabella si selezionano le colonne chiave e le colonne da aggiornare, poi facendo clic su Generate SQL si generano le istruzioni preliminari SQL da associare alle proprietà *ModifySQL*, *InsertSQL* e *DeleteSQL* del componente update. Nella maggior parte dei casi, è preferibile o necessario mettere a punto le istruzioni SQL generate automaticamente.

Per visualizzare e modificare le istruzioni SQL generate, selezionare la pagina SQL. Se sono state generate delle istruzioni SQL, quando la pagina viene selezionata,

l'istruzione relativa alla proprietà *ModifySQL* è già visibile nel riquadro SQL Text. Tale istruzione può essere modificata in base alle prorie esigenze.



Ricordare che le istruzioni SQL generate sono punti di partenza di creare le istruzioni di aggiornamento. In certi casi, bisogna modificarle per farle eseguire correttamente. Ad esempio, quando si gestiscono dati contenenti valori NULL, è necessario modificare la clausola WHERE in modo che sia

```
WHERE field IS NULL
```

invece di usare la variabile di campo generata. Ognuna delle istruzioni va testata personalmente prima di essere accettata.

Per commutare tra le istruzioni SQL generate e modificarle in base alle esigenze, utilizzare i pulsanti di scelta Statement Type.

Per accettare le istruzioni e associarle alle proprietà SQL del componente update, fare clic su OK.

Sostituzione dei parametri nelle istruzioni di aggiornamento SQL

Le istruzioni di aggiornamento SQL ricorrono a una forma particolare di sostituzione di parametro che permette di sostituire vecchi o nuovi valori di campo negli aggiornamenti dei record. L'editor Update SQL quando genera le sue istruzioni, determina quali valori di campo utilizzare. Quando si scrive l'aggiornamento SQL, si specificano i valori del campo da utilizzare.

Quando il nome del parametro corrisponde a un nome di colonna nella tabella, viene automaticamente utilizzato come valore del parametro il nuovo valore del campo nell'aggiornamento presente nella cache del record. Quando il nome del parametro corrisponde a quello di una colonna ed è preceduto dalla stringa "OLD_", viene usato il vecchio valore del campo. Ad esempio, nell'istruzione di aggiornamento SQL qui di seguito, il parametro :LastName viene riempito automaticamente con il nuovo valore del campo presente nell'aggiornamento in cache per il record inserito.

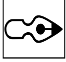
```
INSERT INTO Names
(LastName, FirstName, Address, City, State, Zip)
VALUES (:LastName, :FirstName, :Address, :City, :State, :Zip)
```

Nuovi valori di campo vengono solitamente usati nelle istruzioni *InsertSQL* e *ModifySQL*. Nell'aggiornamento di un record modificato, il nuovo valore di campo preso dalla cache viene utilizzato dall'istruzione UPDATE per sostituire il vecchio valore di campo nella tabella base aggiornata.

Nel caso di record cancellati, non c'è nessun nuovo valore, cosicché la proprietà *DeleteSQL* utilizza la sintassi ":OLD_FieldName". I vecchi valori del campo vengono normalmente usati anche nella clausola WHERE dell'istruzione SQL per gli aggiornamenti di modifica o di cancellazione, in modo da stabilire quali record aggiornare o cancellare.

Nella clausola WHERE di un'istruzione UPDATE o DELETE bisogna fornire almeno il numero minimo di parametri per identificare univocamente il record nella tabella base da aggiornare con i dati della cache. Per esempio, in una lista di clienti, il solo cognome di un cliente potrebbe non essere sufficiente a identificare univocamente nella tabella base il record corretto; potrebbero esserci molti record che hanno

"Smith" come cognome. Invece, i parametri del cognome, del nome e del numero di telefono potrebbero rappresentare una combinazione univoca. Meglio ancora sarebbe un valore univoco del campo come l'identificativo cliente.



Se si creano istruzioni SQL contenenti parametri che non rimandano i valori del campo modificati oppure originali, l'oggetto update non sa come collegare i loro valori. Tuttavia, lo si può fare manualmente, mediante la proprietà *Query* dell'oggetto update. Per i dettagli fare riferimento a "Utilizzo della proprietà Query di un componente update" a pagina 24-49.

Stesura di istruzioni SQL di aggiornamento

In fase di progettazione, l'editor Update SQL può servire a scrivere le istruzioni SQL per le proprietà *DeleteSQL*, *InsertSQL* e *ModifySQL*. Se non lo si usa o se si desidera modificare le istruzioni generate, è bene aver presenti questi criteri quando si scrivono le istruzioni per cancellare, inserire e modificare i record nella tabella base.

La proprietà *DeleteSQL* deve contenere solo un'istruzione SQL con il comando DELETE. La tabella base da aggiornare deve essere indicata nella clausola FROM. Per fare in modo che l'istruzione SQL cancelli nella tabella di base solo il record che corrisponde al record cancellato nella cache di aggiornamento, si utilizza una clausola WHERE. Nella clausola WHERE, si usa un parametro relativo a uno o più campi che consenta di identificare univocamente nella tabella di base il record corrispondente al record da aggiornare nella cache. Se i parametri hanno lo stesso nome del campo e sono preceduti da "OLD_", assumono automaticamente i valori dal campo corrispondente del record da aggiornare memorizzato nella cache. Se i parametri hanno un qualsiasi altro nome, è necessario fornirne i valori.

```
DELETE FROM Inventory I
WHERE (I.ItemNo = :OLD_ItemNo)
```

Alcuni tipi di tabella potrebbero non essere in grado di trovare il record nella tabella base, quando i campi utilizzati per identificare il record contengono valori NULL. In questi casi, l'aggiornamento per cancellazione non va a buon fine. Per ovviare a questa situazione, bisogna aggiungere una condizione ai campi che potrebbero contenere valori NULL utilizzando il predicato IS NULL (oltre a una condizione per i valori non-NULL). Ad esempio, quando un campo FirstName può contenere un valore NULL:

```
DELETE FROM Names
WHERE (LastName = :OLD_LastName) AND
  ((FirstName = :OLD_FirstName) OR (FirstName IS NULL))
```

L'istruzione *InsertSQL* deve contenere solo un'istruzione SQL con il comando INSERT. La tabella base da aggiornare deve essere indicata nella clausola INTO. Nella clausola VALUES, va fornita una lista di parametri separati da virgole. Se i parametri hanno lo stesso nome del campo, ricevono automaticamente il valore del record di aggiornamento memorizzato nella cache. Se i nomi sono diversi, bisogna fornire i valori del parametro. La lista di parametri fornisce i valori dei campi del record appena inserito. Il numero di parametri deve essere uguale a quello dei campi elencati nell'istruzione.

```
INSERT INTO Inventory
(ItemNo, Amount)
```

```
VALUES (:ItemNo, 0)
```

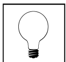
Nell'istruzione *ModifySQL* ci deve essere solo un'istruzione SQL con il comando UPDATE. La tabella base da aggiornare va chiamata nella clausola FROM. Nella clausola SET si includono una o più assegnazioni di valore. Se i valori presenti nella clausola SET sono parametri il cui nome è uguale a quello dei campi, i parametri ricevono automaticamente i valori dei campi con lo stesso nome presenti nel record aggiornato nella cache. Si possono assegnare ulteriori valori di campo mediante altri parametri, purché i nomi siano diversi e i valori siano indicati manualmente. Come con l'istruzione *DeleteSQL*, va fornita una clausola WHERE per verificare univocamente il record da aggiornare utilizzando parametri con lo stesso nome dei campi ma preceduti da "OLD_". Nell'istruzione di aggiornamento qui di seguito, il parametro :ItemNo riceve automaticamente un valore mentre :Price non lo riceve.

```
UPDATE Inventory I
SET I.ItemNo = :ItemNo, Amount = :Price
WHERE (I.ItemNo = :OLD_ItemNo)
```

Considerando l'aggiornamento SQL precedente, ecco un esempio dove l'utente modifica un record esistente. Il valore originale del campo ItemNo è 999. In una griglia connessa al dataset memorizzato nella cache, l'utente fa diventare 123 il valore del campo ItemNo 123 mentre Amount diventa 20. Quando si richiama il metodo ApplyUpdates, questa istruzione SQL agisce su tutti i record della tabella base in cui il campo ItemNo è 999, utilizzando il vecchio valore del campo nel parametro :OLD_itemno. Nei record interessati, cambia il valore del campo ItemNo in 123 (utilizzando il parametro :ItemNo, il valore preso dalla griglia) mentre Amount diventa 20.

Utilizzo di più oggetti update

Quando si devono aggiornare più tabelle base referenziate nel dataset, bisogna usare più oggetti update: uno per ciascuna tabella base aggiornata. Poiché *UpdateObject* del componente dataset consente di associare al dataset un solo oggetto update, è necessario associare a ciascun oggetto un dataset, assegnando alla sua proprietà *DataSet* il nome del dataset.



Quando si utilizzano più oggetti update, è possibile usare *TBDEClientDataSet* invece di *TClientDataSet* con un provider esterno. Questo perché non si deve impostare la proprietà *UpdateObject* del dataset sorgente.

In fase di progetto, nell'Object Inspector non è disponibile la proprietà *DataSet* degli oggetti update. La si può impostare solo in fase di esecuzione.

```
UpdateSQL1->DataSet = Query1;
```

L'oggetto update utilizza questo dataset per ottenere i valori di campi originali e aggiornati per la sostituzione di parametro e, se si tratta di un dataset per BDE, per verificare la sessione e il database da usare quando si trascrivono gli aggiornamenti. Per far funzionare correttamente la sostituzione di parametro, la proprietà *DataSet* dell'oggetto update deve essere il dataset contenente i valori di campo aggiornati. Se si usa un dataset per BDE per gli aggiornamenti in cache, questo è lo stesso dataset per BDE. Se si usa un dataset client, questo è un dataset client fornito come parametro al gestore di evento *BeforeUpdateRecord*.

Quando l'oggetto update non è stato assegnato alla proprietà *UpdateObject* del dataset, le sue istruzioni SQL non vengono eseguite automaticamente nel momento in cui si richiama *ApplyUpdates*. Per aggiornare i record, bisogna chiamare manualmente l'oggetto update da un gestore di evento *OnUpdateRecord* (se è BDE che memorizza gli aggiornamenti nella cache) o da un gestore di evento *BeforeUpdateRecord* (se si usa un dataset client). Nel gestore di evento, le azioni minime da eseguire sono

- Se si utilizza un dataset client per memorizzare nella cache gli aggiornamenti, bisogna accertarsi che le proprietà *DatabaseName* e *SessionName* dell'oggetto da aggiornare siano impostate sui valori delle proprietà *DatabaseName* e *SessionName* del dataset sorgente.
- Il gestore di evento deve chiamare il metodo *ExecSQL* o *Apply* dell'oggetto update. In questo modo, si richiama l'oggetto update per ciascun record da aggiornare. Per ulteriori informazioni sull'esecuzione di istruzioni di aggiornamento, consultare "Esecuzione delle istruzioni SQL" qui di seguito.
- Impostare il parametro *UpdateAction* del gestore di evento a *uaApplied* (*OnUpdateRecord*) oppure assegnare il valore **true** al parametro *Applied* (*BeforeUpdateRecord*).

Facoltativamente, si possono convalidare o modificare i dati, oppure effettuare altre operazioni che dipendono dall'aggiornamento di ciascun record.



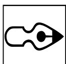
Se in un gestore di evento *OnUpdateRecord* si chiama il metodo *ExecSQL* o *Apply* di un oggetto update, accertarsi di non impostare la proprietà *UpdateObject* del dataset a quell'oggetto update, altrimenti seguirà un secondo tentativo di aggiornare ogni record.

Esecuzione delle istruzioni SQL

Quando si usano più oggetti update, non si deve associarli a un dataset impostandone la proprietà *UpdateObject*. La conseguenza sarebbe che le istruzioni appropriate non verrebbero eseguite automaticamente al momento di applicare gli aggiornamenti. Invece, l'oggetto update va richiamato esplicitamente scrivendo un codice.

Ci sono due modi per richiamare l'oggetto update. Il modo scelto dipende dal fatto se l'istruzione SQL utilizza o meno dei parametri per rappresentare i valori del campo:

- Se l'istruzione SQL da eseguire utilizza parametri, chiamare il metodo *Apply*.
- Se l'istruzione SQL da eseguire non utilizza parametri, è meglio chiamare il metodo *ExecSQL*.



Se l'istruzione SQL utilizza parametri diverso dai tipi incorporati (sia per i valori del campo originali che per quelli aggiornati), i valori del parametro vanno forniti manualmente invece di basarsi sulla sostituzione di parametro fornita dal metodo *Apply*. Per informazioni su come indicare manualmente i valori dei parametri, vedere "Utilizzo della proprietà Query di un componente update" a pagina 24-49.

Per informazioni sulla sostituzione dei parametri predefiniti nelle istruzioni SQL di un oggetto update, vedere "Sostituzione dei parametri nelle istruzioni di aggiornamento SQL" a pagina 24-44.

Chiamata del metodo Apply

Il metodo *Apply* di un componente update applica manualmente gli aggiornamenti al record attivo. Il rpocedimento si svolge in due fasi:

1. I valori iniziale e modificato del campo del record sono collegati a parametri nell'istruzione SQL appropriata.
2. L'istruzione SQL viene eseguita.

Per applicare l'aggiornamento del record attivo nella cache di aggiornamento chiamare il metodo *Apply*. Il metodo *Apply* è chiamato molto spesso dall'interno di un gestore di evento *OnUpdateRecord* del dataset o dal gestore di evento *BeforeUpdateRecord* di un provider.



Se si utilizza la proprietà *UpdateObject* del dataset per associare il dataset e l'oggetto update, *Apply* viene chiamato automaticamente. In questo caso, non bisogna chiamarlo in un gestore di evento *OnUpdateRecord* in quanto vi sarebbe un secondo tentativo di applicare l'aggiornamento del record attivo.

I gestori di evento *OnUpdateRecord* indicano il tipo di aggiornamento da applicare con un parametro *UpdateKind* di tipo *TUpdateKind*. Il parametro va passato al metodo *Apply* per indicare quale istruzione SQL utilizzare. Il seguente codice lo illustra utilizzando un gestore di evento *BeforeUpdateRecord*:

```
void __fastcall TForm1::BDEClientDataSet1BeforeUpdateRecord(TObject *Sender,
   TDataSet *SourceDS, TCustomClientDataSet *DeltaDS, TUpdateKind UpdateKind, bool &Applied)
{
  UpdateSQL1->DataSet = DeltaDS;
  TDBDataSet *pSrcDS = dynamic_cast<TDBDataSet *>(SourceDS);
  UpdateSQL1->DatabaseName = pSrcDS->DatabaseName;
  UpdateSQL1->SessionName = pSrcDS->SessionName;
  UpdateSQL1->Apply(UpdateKind);
  Applied = true;
}
```

Chiamata del metodo ExecSQL

Il metodo *ExecSQL* di un componente update applica manualmente gli aggiornamenti al record attivo. A differenza del metodo *Apply*, *ExecSQL* non associa dei parametri all'istruzione SQL prima di eseguirla. Il metodo *ExecSQL* è chiamato molto spesso dall'interno di un gestore di evento *OnUpdateRecord* (quando si usa BDE), oppure *BeforeUpdateRecord* (quando si usa un dataset client).

Siccome *ExecSQL* non collega valori di parametro, è usato soprattutto quando le istruzioni SQL dell'oggetto update non contengono parametri. In assenza di parametri, si può usare anche *Apply*, ma *ExecSQL* è più efficiente perché non fa la verifica dei parametri.

Se le istruzioni SQL includono parametri, si può sempre chiamare *ExecSQL,* ma solo dopo aver collegato esplicitamente i parametri. Se si utilizza BDE per mettere nella cache gli aggiornamenti, è possibile collegare esplicitamente i parametri impostando la proprietà *DataSet* dell'oggetto update e poi chiamando il suo metodo *SetParams*. Quando si utilizza un dataset client per memorizzare nella cache gli aggiornamenti, è necessario fornire i parametri al relativo oggetto query gestito da *TUpdateSQL*. Per

informazioni, consultare “Utilizzo della proprietà Query di un componente update” a pagina 24-49.



Se la proprietà *UpdateObject* del dataset viene usata per associare dataset e oggetto update, *ExecSQL* viene chiamato automaticamente. In questo caso, non si deve chiamare *ExecSQL* in un gestore di evento *OnUpdateRecord* o *BeforeUpdateRecord*, in quanto si avrebbe un secondo tentativo di applicare l’aggiornamento al record attivo.

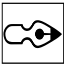
I gestori di evento *OnUpdateRecord* e *BeforeUpdateRecord* indicano il tipo di aggiornamento da applicare con un parametro *UpdateKind* di tipo *TUpdateKind*. Il parametro va passato al metodo *ExecSQL* per indicare quali istruzioni SQL utilizzare. Il seguente codice lo illustra utilizzando un gestore di evento *BeforeUpdateRecord*:

```
void __fastcall TForm1::BDEClientDataSet1BeforeUpdateRecord(TObject *Sender,
   TDataSet *SourceDS, TCustomClientDataSet *DeltaDS, TUpdateKind UpdateKind, bool &Applied)
{
  TDBDataSet *pSrcDS = dynamic_cast<TDBDataSet *>(SourceDS);
  UpdateSQL1->DatabaseName = pSrcDS->DatabaseName;
  UpdateSQL1->SessionName = pSrcDS->SessionName;
  UpdateSQL1->ExecSQL(UpdateKind);
  Applied = true;
}
```

Se durante l’esecuzione del programma di aggiornamento viene sollevata un’eccezione, l’esecuzione continua nell’evento *OnUpdateError*, se questo è definito.

Utilizzo della proprietà Query di un componente update

La proprietà *Query* di un componente update consente l’accesso ai componenti query che implementano le sue istruzioni *DeleteSQL*, *InsertSQL* e *ModifySQL*. Nella maggior parte delle applicazioni, non è necessario accedere direttamente a questi componenti query. Si specificano le istruzioni che le query eseguono chiamando il metodo *Apply* o *ExecSQL* dell’oggetto update, mediante le proprietà *DeleteSQL*, *InsertSQL* e *ModifySQL*. In certi casi però è necessario manipolare direttamente un componente query. In particolare, la proprietà *Query* è utile quando si desidera fornire i valori dei parametri nelle istruzioni SQL, invece di affidarsi al collegamento automatico dei parametri dell’oggetto update ai valori vecchi e nuovi del campo.



La proprietà *Query* è accessibile solo in fase di esecuzione.

La proprietà *Query* è associata a indice su un valore di *TUpdateKind*:

- L’utilizzo di un indice di *ukModify* permette di accedere alla query che aggiorna record esistenti.
- L’utilizzo di un indice di *ukInsert* permette di accedere alla query che inserisce nuovi record.
- L’utilizzo di un indice di *ukDelete* permette di accedere alla query che cancella record.

Quanto segue mostra come utilizzare la proprietà *Query* per fornire valori del parametro che non possono essere collegati automaticamente:

```
void __fastcall TForm1::BDEClientDataSet1BeforeUpdateRecord(TObject *Sender,
   TDataSet *SourceDS, TCustomClientDataSet *DeltaDS, TUpdateKind UpdateKind, bool &Applied)
{
  UpdateSQL1->DataSet = DeltaDS; // required for the automatic parameter substitution
```

```
    TQuery *pQuery = UpdateSQL1->Query[UpdateKind]; // access the query
    // make sure the query has the correct DatabaseName and SessionName
    TDBDataSet *pSrcDS = dynamic_cast<TDBDataSet *>(SourceDS);
    pQuery->DatabaseName = pSrcDS->DatabaseName;
    pQuery->SessionName = pSrcDS->SessionName;
    // now substitute values for custom parameters
    pQuery->ParamByName("TimeOfLastUpdate")->Value = Now();
    UpdateSQL1->Apply(UpdateKind); // now do automatic substitution and execute
    Applied = true;
}
```

Utilizzo di TBatchMove

TBatchMove incorpora alcune funzioni del Borland Database Engine (BDE) che consentono di duplicare un dataset, di aggiungere a un dataset record presi da un altro, di aggiornare i record di un dataset con record presi da un altro dataset e di cancellare da un dataset record corrispondenti a record di un altro dataset. *TBatchMove* viene usato principalmente per:

- Trasferire dati da un server a un datasource locale per analizzarli o per svolgere altre operazioni.
- Spostare un database desktop in tabelle su un server remoto come parte di un'operazione di upsizing.

Un componente batch move può creare sulla destinazione tabelle corrispondenti alle tabelle sorgente, mappando automaticamente i nomi di colonna e i tipi di dati, secondo quanto opportuno.

Creazione di un componente batch move

Per creare un componente batch move:

1. Mettere su una scheda o in un modulo dati un componente tabella o query per il dataset da cui si vogliono importare record (chiamato dataset *Source*).
2. Mettere nella scheda o nel modulo dati il dataset dove si vogliono spostare i record (chiamato dataset *Destination*).
3. Mettere nel modulo dati o nella scheda un componente *TBatchMove* prendendolo dalla pagina BDE della Component palette e dare alla sua proprietà *Name* un valore univoco, appropriato all'applicazione.
4. Impostare la proprietà *Source* del componente batch move al nome della tabella dove copiare, aggiungere o aggiornare i record. È possibile selezionare le tabelle desiderate nella lista a discesa dei componenti dataset disponibili.
5. Impostare la proprietà *Destination* del dataset dove creare, aggiungere a o aggiornare. Si può selezionare una tabella di destinazione nella lista a discesa dei componenti dataset disponibili.

- In caso di aggiunta, aggiornamento o cancellazione, *Destination* deve essere una tabella di database esistente.
- Se si vuole copiare una tabella e *Destination* è una tabella esistente, eseguendo lo spostamento batch vengono sovrascritti tutti i dati correnti nella tabella di destinazione.
- Se si crea una tabella nuova copiando una tabella esistente, il nome della tabella risultante è quello specificato dalla proprietà *Name* del componente tabella dal quale si copia. Il tipo di struttura della tabella risultante sarà appropriato al server specificato dalla proprietà *DatabaseName*.

6 Impostare la proprietà *Mode* per indicare il tipo di operazione da svolgere. Le operazioni valide sono *batAppend* (l'impostazione predefinita), *batUpdate*, *batAppendUpdate*, *batCopy* e *batDelete*. Per informazioni su questi modi, consultare "Specifica della modalità di un componente batch move" a pagina 24-51.

7 Facoltativamente, impostare la proprietà *Transliterate*. Se *Transliterate* è **true** (l'impostazione predefinita), i dati di tipo carattere sono tradotti dal set di caratteri del dataset *Source* al set di caratteri del dataset *Destination*, secondo quanto opportuno.

8 Facoltativamente, impostare le mappature di colonna utilizzando la proprietà *Mappings*. Non è necessario impostare la proprietà se si desidera che nello spostamento batch la corrispondenza tra le colonne sia basata sulla rispettiva posizione nelle tabelle di origine e di destinazione. Per ulteriori informazioni sulla mappatura delle colonne, vedere "Mappatura dei tipi di dati" a pagina 24-53.

9 Facoltativamente, specificare le proprietà *ChangedTableName*, *KeyViolTableName* e *ProblemTableName*. Durante lo spostamento batch i record che creano problemi incontrati nel corso dell'operazione vengono messi nella tabella specificata da *ProblemTableName*. Quando si aggiorna una tabella Paradox mediante uno spostamento batch, le violazioni di chiave si possono segnalare nella tabella specificata in *KeyViolTableName*. *ChangedTableName* elenca tutti i record che sono cambiati nella tabella di destinazione come conseguenza dell'operazione di spostamento batch. Non specificando queste proprietà , le tabelle di errore non vengono create né utilizzate. Per ulteriori informazioni sulla gestione di errori di spostamento batch, consultare "Gestione degli errori delle operazioni batch move" a pagina 24-54.

Specifica della modalità di un componente batch move

La proprietà *Mode* specifica l'operazione che un componente batch move esegue:

Tabella 24.8 Modalità di un componente batch move

Proprietà	Scopo
batAppend	Aggiunge record alla tabella di destinazione.
batUpdate	Aggiorna i record nella tabella di destinazione con record corrispondenti presi nella tabella di origine. L'aggiornamento si basa sull'indice ativo della tabella di destinazione.

Tabella 24.8 Modalità di un componente batch move (continua)

Proprietà	Scopo
batAppendUpdate	Se nella tabella di destinazione esiste un record corrispondente, lo aggiorna. Altrimenti, aggiunge i record nella tabella di destinazione.
batCopy	Crea la tabella di destinazione in base alla struttura della tabella di origine. Se la tabella di destinazione esiste già , viene accantonata e ricreata.
batDelete	Cancella nella tabella di destinazione i record corrispondenti ai record nella tabella di origine.

Aggiunta di record

Per l'aggiunta di dati, il dataset di destinazione deve rappresentare una tabella esistente. Durante l'operazione di aggiunta, BDE converte i dati per adeguarli ai tipi e alle dimensioni del dataset di destinazione, se necessario. Se la conversione non è possibile, viene sollevata un'eccezione e i dati non vengono aggiunti.

Aggiornamento di record

Per l'aggiornamento dei dati, il dataset di destinazione deve rappresentare una tabella esistente e deve avere un indice definito che consenta il confronto dei record. Se per il confronto sono usati i campi indice primari, i record che nel dataset di destinazione hanno campi indice corrispondenti ai campi indice dei record nel dataset di origine vengono riscritti con i dati del dataset di origine. Durante l'operazione di aggiornamento, BDE converte i dati per adeguarli ai tipi e alle dimensioni del dataset di destinazione, se necessario.

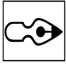
Aggiunta e aggiornamento di record

Per aggiungere e aggiornare dati, il dataset di destinazione deve rappresentare una tabella esistente e deve avere un indice definito che consenta di confrontare i record. Se per il confronto si utilizzano i campi dell'indice principale, i record che nel dataset destinazione hanno campi indice corrispondenti ai record del dataset di origine vengono sovrascritti con i dati del dataset sorgente. In caso contrario, al dataset destinazione vengono aggiunti i dati presi dal dataset sorgente. Durante l'operazione di aggiunta e di aggiornamento, BDE converte i dati per adeguarli ai tipi e alle dimensioni del dataset di destinazione, se necessario.

Copia di dataset

Per copiare un dataset sorgente, il dataset di destinazione non deve rappresentare una tabella esistente. In caso contrario, l'operazione di spostamento batch sovrascrive la tabella esistente con una copia del dataset di origine.

Se i dataset di origine e di destinazione sono gestiti da motori di database di tipo diverso, (per esempio, Paradox e InterBase), BDE crea a destinazione un dataset la cui struttura è quanto più possibile simile a quella del dataset sorgente ed esegue automaticamente le conversioni di tipo e di dimensione dei dati, secondo quanto necessario.



TBatchMove non copia strutture di metadati come indici, vincoli e procedure registrate. È necessario ricreare questi oggetti di metadati sul server di database, o attraverso lo SQL Explorer, secondo quanto necessario.

Cancellazione di record

Per cancellare dati nel dataset di destinazione, questo deve rappresentare una tabella esistente e deve avere un indice definito che consenta il confronto dei record. Se per il confronto si usano i campi dell'indice principale, i record i cui campi indice nel dataset destinazione corrispondono ai campi indice dei record nel dataset sorgente vengono cancellati nella tabella di destinazione.

Mappatura dei tipi di dati

In modalità *batAppend*, un componente batch move crea la tabella di destinazione in base ai tipi di dati della colonna della tabella di origine. Le colonne e i tipi vengono confrontati in base alla rispettiva posizione nelle tabelle di origine e di destinazione. In altre parole, la prima colonna della sorgente viene fatta corrispondere con la prima colonna della destinazione, e così via.

Per ridefinire le mappature di colonna predefinite, si ricorre alla proprietà *Mappings*. *Mappings* è una lista di mappature di colonna (una per riga) che può assumere due forme. Per mappare una colonna della tabella sorgente con una colonna dallo stesso nome nella tabella di destinazione, si può usare un listato semplice che specifica il nome della colonna da far corrispondere. Ad esempio, la seguente mappatura specifica che una colonna chiamata *ColName* nella tabella sorgente va fatta corrispondere con una colonna dello stesso nome nella tabella di destinazione:

```
ColName
```

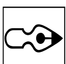
Per fare corrispondere una colonna chiamata *SourceColName* nella tabella sorgente con una colonna chiamata *DestColName* nella tabella di destinazione, la sintassi è :

```
DestColName = SourceColName
```

Se i tipi di dati della colonna sorgente e di destinazione non sono uguali, l'operazione di spostamento batch prova ad applicare la ″migliore delle corrispondenze″. Tronca i dati di tipo carattere, se necessario, e tenta un po' di conversione, se possibile. (Ad esempio, mappando una colonna CHAR(10) verso una colonna CHAR(5) troncherà gli ultimi cinque caratteri della colonna sorgente.

Come esempio di conversione, se una colonna sorgente di dati di tipo carattere viene fatta corrispondere con una colonna di destinazione di tipo integer, l'operazione di spostamento batch converte il carattere ′5 ′ nel corrispondente valore integer. Valori che non possono essere convertiti generano errori. Per ulteriori informazioni sugli errori, consultare "Gestione degli errori delle operazioni batch move" a pagina 24-54.

Quando sposta dati fra tabelle di tipo diverso, un componente batch move traduce opportunamente i tipi di dati in base al tipo di server del dataset. Per conoscere le più recenti tabelle di mappatura tra i vati tipi server, consultare la guida BDE in linea.



Per gli spostamenti batch verso server di database SQL, bisogna avere il server di database in questione e una versione di C++Builder sulla quale sia installato lo SQL

Link appropriato, oppure si può usare ODBC, se si dispone degli opportuni driver ODBC.

Esecuzione di un'operazione batch move

In fase di esecuzione, per eseguire un'operazione batch precedentemente preparata si utilizza il metodo *Execute.* Ad esempio, se *BatchMoveAdd* è il nome di un componente batch move, la seguente istruzione lo esegue:

```
BatchMoveAdd->Execute();
```

Si può anche eseguire uno spostamento batch in fase di progettazione facendo clic con il pulsante destro su un componente batch move e poi selezionando Execute nel menu di scelta rapida.

Negli spostamenti batch, la proprietà *MovedCount* conteggia il numero di record spostati.

La proprietà *RecordCount* specifica il numero massimo di record da spostare. Se *RecordCount* è nullo, tutti i record vengono spostati, iniziando con il primo record nel dataset sorgente. Se *RecordCount* è un numero positivo, vengono spostati i record fino a un massimo di *RecordCount*, iniziando dal record attivo nel dataset sorgente. Se *RecordCount* è maggiore del numero di record tra il record attivo nel dataset sorgente e il suo ultimo record, lo spostamento batch termina quando si raggiunge la fine del dataset sorgente. È possibile esaminare *MoveCount* per determinare quanti record sono stati effettivamente trasferiti.

Gestione degli errori delle operazioni batch move

Due tipi di errori possono verificarsi nelle operazioni batch move: gli errori di conversione del tipo di dati e le violazioni di integrità . *TBatchMove* ha diverse proprietà per rilevare e gestire gli errori.

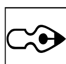
La proprietà *AbortOnProblem* specifica se l'operazione va interrotta quando si verifica un errore di conversione di tipo dati. Se *AbortOnProblem* è **true**, l'operazione batch move viene annullata quando si verifica un errore. Se **false**, l'operazione continua. È possibile esaminare la tabella che si specifica in *ProblemTableName* per determinare quali record hanno causato problemi.

La proprietà *AbortOnKeyViol* indica se l'operazione va interrotta quando si verifica una violazione di chiave Paradox.

La proprietà *ProblemCount* indica il numero di record che non hanno potuto essere gestiti nella tabella di destinazione senza perdite di dati. Se *AbortOnProblem* è **true**, questo numero è uno, poiché l'operazione viene interrotta non appena si verifica un errore.

Le seguenti proprietà permettono a un componente batch move di creare tabelle aggiuntive che documentano l'operazione di spostamento batch:

- *ChangedTableName*, se specificato, crea una tabella locale Paradox contenente tutti i record nella tabella di destinazione che sono cambiati a seguito di un'operazione di aggiornamento o di cancellazione.
- *KeyViolTableName,* se specificato, crea una tabella locale Paradox che contiene tutti i record dalla tabella sorgente che hanno determinato una violazione di chiave quando lavoravano con una tabella Paradox. Se *AbortOnKeyViol* è **true**, la tabella conterrà al massimo un voce poiché l'operazione viene interrotta al primo problema incontrato.
- *ProblemTableName*, se specificato, crea una tabella locale Paradox contenente tutti i record che non è stato possibile registrare nella tabella di destinazione a causa di errori di conversione del tipo dati. Ad esempio, la tabella potrebbe contenere i record dalla tabella di sorgente contenenti dati che, per essere inseriti nella tabella di destinazione, dovevano essere troncati. Se *AbortOnProblem* è **true**, la tabella contiene al massimo un record poiché l'operazione è stata interrotta al primo problema incontrato.



Non specificando *ProblemTableName*, i dati nel record vengono troncati e inseriti nella tabella di destinazione.

Il Data Dictionary

Quando si utilizza BDE per accedere ai dati, l'applicazione ha accesso al Data Dictionary. Il Data Dictionary mette a disposizione un'area di memoria personalizzabile, indipendente dalle applicazioni, dove è possibile creare set di attributi di campi estesi che descrivono il contenuto e l'aspetto dei dati.

Ad esempio, se si sviluppano frequentemente applicazioni finanziarie, è possibile creare un certo numero di set speciali di attributi di campo che descrivono vari formati di visualizzazione delle valute. Quando i dataset dell'applicazione vengono creati in fase di progetto, invece che impostando manualmente i campi delle valute in ciascun dataset, si possono associare i campi in questione a un set esteso di attributi di campo nel dizionario dati. L'uso del dizionario dati assicura un aspetto coerente dei dati all'interno delle singole applicazioni e tra un'applicazione e l'altra.

In un ambiente client/server, il Data Dictionary può risiedere su un server remoto, in modo da consentire un'ulteriore condivisione delle informazioni.

Per imparare in che modo creare set estesi di attributi di campo utilizzando l'editor Fields in fase di progettazione, e come associarli ai campi di tutti i dataset di un'applicazione, consultare "Creazione di set di attributi per i componenti campo" a pagina 23-14. Per saperne di più sulla creazione di un dizionario dati e degli attributi di campo estesi con SQL e con Database Explorers, consultare i relativi file di guida in linea.

Un'interfaccia di programmazione al Data Dictionary è disponibile nel file header drintf (situato nella directory include\VCL). L'interfaccia fornisce i seguenti metodi:

Tabella 24.9 Interfaccia del Data Dictionary

Routine	Utilizzo
DictionaryActive	Indica se il dizionario dati è attivo.
DictionaryDeactivate	Disattiva il dizionario dati.
IsNullID	Indica se un certo ID è un ID nullo
FindDatabaseID	Restituisce l'ID di un database fornendo il suo alias.
FindTableID	Restituisce l'ID di una tabella in un database specificato.
FindFieldID	Restituisce l'ID di un campo in una tabella specificata.
FindAttrID	Restituisce l'ID di un set di attributi dato.
GetAttrName	Restituisce il nome di un set di attributi dato il suo ID.
GetAttrNames	Esegue una callback per ogni attributo impostato nel dizionario.
GetAttrID	Restituisce l'ID dell'attributo impostato di un campo specificato.
NewAttr	Crea un nuovo set di attributi da un componente campo.
UpdateAttr	Aggiorna un set di attributi corrispondente alle proprietà di un campo.
CreateField	Crea un componente campo in base agli attributi registrati.
UpdateField	Modifica le proprietà di un campo per farle corrispondere a un set di attributi specificato.
AssociateAttr	Associa un set di attributi all'ID specificato di un campo.
UnassociateAttr	Rimuove un set di attributi dall'ID di un campo.
GetControlClass	Restituisce la classe di controllo dell'ID specificato di un attributo.
QualifyTableName	Restituisce un nome di tabella per esteso (qualificato dal nome utente).
QualifyTableNameByName	Restituisce un nome della tabella per esteso (qualificato dal nome utente).
HasConstraints	Indica se il dataset ha vincoli nel dizionario.
UpdateConstraints	Aggiorna i vincoli importati di un dataset.
UpdateDataset	Aggiorna un dataset secondo le impostazioni e i vincoli attivi nel dizionario.

Strumenti per operare con BDE

Uno dei vantaggi di usare BDE come meccanismo di accesso di dati è la ricchezza di programmi di utilità di cui C++Builder dispone. Questi programmi di utilità comprendono:

- **SQL Explorer** e **Database Explorer**: C++Builder viene fornito con una di queste due applicazioni, a seconda della versione acquistata. Entrambi consentono di
  - Esaminare le tabelle e le strutture di database esistenti. SQL Explorer permette di esaminare e di eseguire query su database SQL remoti.

- Riempire tabelle di dati
- Creare set estesi di attributi di campo nel Data Dictionary o associarli ai campi di un'applicazione.
- Creare e gestire alias BDE.

SQL Explorer consente inoltre di:

- Creare oggetti SQL, come procedure registrate su server di database remoti.
- Visualizzare il testo ricostruito di oggetti SQL su server di database remoti.
- Eseguire procedure SQL.

- **SQL Monitor**: SQL Monitor permette di tenere sotto controllo le comunicazioni fra il server di database remoto e BDE. Si possono filtrare i messaggi che si desidera guardare, limitandoli solo alle categorie che interessano. SQL Monitor è molto utile in fase di debug delle applicazioni.
- **Programma di utilità BDE Administration:** Il programma di utilità BDE Administration permette di aggiungere nuovi driver di database, di configurare le impostazioni predefinite dei driver esistenti e di creare nuovi alias BDE.
- **Database Desktop**: Se si utilizzano tabelle Paradox o dBASE, Database Desktop permette di visualizzare e di modificare i dati, di creare nuove tabelle e di ristrutturare le tabelle esistenti. Il controllo consentito da Database Desktop è migliore di quello ottenuto utilizzando i metodi di un componente *TTable* (ad esempio, permette di specificare i controlli di validità e i driver di linguaggio). Inoltre, fornisce l'unico meccanismo per ristrutturare le tabelle Paradox e dBASE che non sia quello di fare chiamate dirette alla API di BDE.

Capitolo

25

Operazioni con componenti ADO

I componenti *dbGo* forniscono l'accesso ai dati tramite il framework ADO. ADO (acronimo di ActiveX Data Objects di Microsoft), è una serie di oggetti COM che consentono l'accesso ai dati tramite un DB provider OLE. I componenti *dbGo* incapsulano questi oggetti ADO nell'architettura di database C++Builder.

Nelle applicazioni basate su ADO, lo strato ADO è formato da Microsoft ADO 2.1, da un DB provider OLE o da un driver ODBC per l'accesso al datastore, dal software client per lo specifico sistema di database utilizzato (nel caso di database *SQL*), da un sistema di database back-end (per sistemi di database *SQL*) e da un database. Perché l'applicazione sia completamente funzionale, tutti questi sistemi devono essere accessibili all'applicazione basata su ADO.

Gli oggetti ADO più importanti sono gli oggetti *Connection*, Command e *Recordset*. Questi oggetti ADO sono rappresentati dai componenti *TADOConnection*, *TADOCommand*, e dai componenti ADO dataset. Il framework ADO include altri oggetti "helper", come gli oggetti Field e Properties, ma solitamente non sono utilizzati in modo diretto nelle applicazioni *dbGo* e non sono inglobati in componenti dedicati.

Questo capitolo presenta i componenti *dbGo* e discute le funzionalità esclusive che essi introducono nella comune architettura database di C++Builder. Prima di affrontare le funzionalità caratteristiche dei componenti *dbGo*, è bene imparare a conoscere le funzionalità comuni dei componenti connection di database e dei dataset descritti nel capitolo Capitolo 21, "Connessione ai database" e nel Capitolo 22, "I dataset."

Panoramica dei componenti ADO

La pagina ADO page della Component palette contiene i componenti *dbGo*. Questi componenti consentono ai programmatori di connettersi ad un datastore ADO e quindi di eseguire comandi e recuperare dati dalle tabelle nel database utilizzando il framework ADO. Sul computer host deve essere installato ADO 2.1 (o una versione

successiva). Inoltre, deve essere installato un software client per il sistema di database di destinazione (come Microsoft *SQL* Server), oltre ad un driver OLE DB o un driver ODBC specifico per quel particolare sistema di database.

La maggior parte dei componenti *dbGo* hanno delle controparti dirette nei componenti disponibili per altri meccanismi di accesso ai dati: un componente connection di database (*TADOConnection*) e diversi tipi di dataset. Inoltre, *dbGo* include *TADOCommand*, un componente semplice che non è un dataset ma che rappresenta un comando *SQL* da eseguire sul datastore ADO.

In tabella sono elencati i componenti ADO.

Tabella 25.1 Componenti ADO

Componente	Uso
TADOConnection	Un componente database connection per stabilire una connessione con un datastore ADO; più componenti dataset e command di ADO possono condividere questa connessione per eseguire comandi, recuperare dati e operare sui metadati.
TADODataSet	Il dataset primario utilizzato per reperire e operare sui dati; *TADODataSet* può recuperare dati da una o più tabelle; può connettersi a un datastore in modo diretto oppure tramite un componente *TADOConnection*.
TADOTable	Un dataset di tipo tabella per reperire e operare su un recordset prodotto da una singola tabella di database; *TADOTable* può connettersi a un datastore in modo diretto oppure tramite un componente *TADOConnection*.
TADOQuery	Un dataset di tipo query per reperire e operare su un recordset prodotto da un'istruzione *SQL* valida; *TADOQuery* può eseguire anche istruzioni *SQL* DDL (data definition language). Può connettersi a un datastore in modo diretto o tramite un componente *TADOConnection*
TADOStoredProc	Un dataset di tipo procedura registrata per eseguire procedure registrate; *TADOStoredProc* esegue procedure registrate che recuperano o non recuperano dati. Può connettersi a un datastore in modo diretto o tramite un componente *TADOConnection*.
TADOCommand	Un semplice componente per eseguire comandi (istruzioni *SQL* che non restituiscono set di risultati); *TADOCommand* può essere utilizzato con un componente dataset di supporto, o recuperare un dataset da una tabella; può connettersi a un datastore in modo diretto o tramite un componente *TADOConnection*.

Connessione a datastore ADO

Le applicazioni *dbGo* utilizzano ActiveX Data Objects (ADO) 2.1 di Microsoft per interagire con un provider OLE DB che si collega a un datastore e accede ai suoi dati. Uno degli elementi che un datastore può rappresentare è un database. Un'applicazione basata su ADO richiede l'installazione di ADO 2.1 sul computer client. ADO e OLE DB sono forniti da Microsoft e sono installati con Windows.

Un provider ADO è uno dei vari tipi possibili di accesso, dai driver nativi OLE DB ai driver ODBC. Questi driver devono essere installati sul computer client. I driver OLE DB dei diversi sistemi di database sono forniti dal produttore del database o da terzi.

Se l'applicazione utilizza un database *SQL*, come Microsoft *SQL* Server o Oracle, il software client del sistema di database deve essere installato anche sul computer client. Il software client è fornito dal produttore del database e viene installato dal CD (o dal disco) del sistema di database).

Per collegare l'applicazione al datastore, si utilizza un componente connection ADO (*TADOConnection*), che va configurato per l'impiego di uno dei provider ADO disponibili. Configurare il componente ADO connection in modo che utilizzi uno dei provider ADO disponibili. Benché *TADOConnection* non sia strettamente necessario, dato che il comando e i componenti dataset ADO possono connettersi utilizzando direttamente la tloro proprietà *ConnectionString*, è possibile utilizzarlo per condividere una sola connessione tra più componenti ADO. In questo modo, si può ridurre la quantità di risorse utilizzate e si ha la possibilità di creare transazioni che attraversano più dataset.

Come altri componenti di connessione ai database, *TADOConnection* supporta queste funzioni

- Controllo delle connessioni
- Controllo del login al server
- Gestione delle transazioni
- Operazioni con i dataset associati
- Invio di comandi al server
- Ottenimento di metadati

Oltre a queste funzionalità , comuni a tutti i componenti di connessione ai database, *TADOConnection* supporta

- Un'ampia gamma di opzioni utilizzabili per la messa a punto della connessione.
- La capacità di elencare gli oggetti command che utilizzano la connessione.
- Eventi ulteriori quando vengono eseguite operazioni comuni.

Connessione a un datastore mediante TADOConnection

Uno o più componenti dataset e comandi ADO possono condividere una singola connessione a un datastore mediante l'utilizzo di *TADOConnection*. A questo scopo i componenti dataset e comandi vengono associati al componente connessione mediante le rispettive proprietà *Connection*. In fase di progetto, selezionare nell'Object Inspector il componente connessione desiderato dall'elenco a discesa della proprietà *Connection*. In fase di esecuzione, assegnare il riferimento alla proprietà *Connection*. Per esempio, il comando seguente associa un componente *TADODataSet* con un componente *TADOConnection*.

```
ADODataSet1->Connection = ADOConnection1;
```

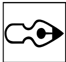
Il componente connection rappresenta un oggetto di connessione ADO. Prima di poterlo utilizzare per stabilire una connessione, è necessario identificare il datastore a cui ci si vuole collegare. Di solito, si forniscono informazioni mediante la proprietà *ConnectionString*, una stringa delimitata da punto e virgola che elenca uno o più parametri di connessione indicati con nome. *ConnectionString* è una stringa separata con punti e virgola che elenca uno o più parametri di connessione con nome. Questi parametri identificano il datastore specificando o il nome di un file contenente le

informazioni di connessione, o il nome di un provider ADO e un riferimento che identifica il datastore. Per fornire le informazioni, si utilizzano i seguenti i nomi predefiniti di parametro:

Parametro	Descrizione
Provider	Nome del provider locale ADO da utilizzare per la connessione.
Data Source	Nome del datastore.
File name	Nome del file contenente informazioni di connessione.
Remote Provider	Nome del provider ADO residente su una macchina remota.
Remote Server	Nome del server remoto quando si utilizza un provider remoto.

Così, un valore tipico di *ConnectionString* si presenta in questo modo

```
Provider=MSDASQL.1;Data Source=MQIS
```

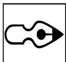


I parametri di connessione di *ConnectionString* non devono obbligatoriamente includere il parametro *Provider*, o *Remote Provider*, se il provider ADO viene specificato utilizzando la proprietà *Provider*. Analogamente, non bisogna specificare il parametro *Data Source* se si utilizza la proprietà *DefaultDatabase*.

Inoltre, ai parametri elencati, *ConnectionString* può aggiungere particolari parametri di connessione richiesti dallo specifico provider ADO utilizzato. Questi parametri di connessione aggiuntivi possono includere l'ID utente e la password, se si preferisce immettere mediante codice le informazioni di login.

In progettazione, è possibile utilizzare il Connection String Editor per generare una stringa di connessione selezionando da elenchi gli elementi della connessione (come il provider e il server). Fare clic nell'Object Inspector sul pulsante ellissi della proprietà *ConnectionString* per attivare il Connection String Editor, che è un editor di proprietà ActiveX fornito da ADO.

Una volta specificata la proprietà *ConnectionString* (e, facoltativamente, la proprietà *Provider*), si può utilizzare il componente connection ADO per connettersi al datastore ADO o scollegarsi, sebbene in certi casi si preferisca prima utilizzare altre proprietà per mettere a punto la connessione. Quando si collega o si scollega dal datastore, *TADOConnection* permette di rispondere ad alcuni eventi ulteriori, oltre quelli comuni a tutti i componenti connection database. Questi eventi aggiuntivi sono descritti in "Eventi allo stabilirsi di una connessione" a pagina 25-8 e "Eventi alla chiusura del collegamento" a pagina 25-8.



Se non si attiva esplicitamente la connessione impostando a **true** la proprietà *Connected* del componente connessione, l'attivazione avverrà automaticamente non appena si attiverà il primo componente dataset oppure si eseguirà per la prima volta un comando usando un componente comandi ADO.

Accesso all'oggetto connessione

Utilizzare la proprietà *ConnectionObject* di *TADOConnection* per accedere direttamente all'oggetto connessione ADO sottostante. Utilizzando questo riferimento, è possibile accedere alle proprietà e chiamare i metodi dell'oggetto ADO *Connection* sottostante.

L'uso dell'oggetto ADO *Connection* sottostante richiede una buona conoscenza operativa degli oggetti ADO in generale e delloggetto ADO *Connection* in particolare. Non si consiglia l'utilizzo dell'oggetto *Connection* a meno che non si abbia familiarità con le operazioni con gli oggetti *Connection*. Per informazioni specifiche sull'utilizzo degli oggetti *Connection* ADO, consultare la Guida di Microsoft Data Access SDK.

Messa a punto di una connessione

Uno dei vantaggi dell'impiego di *TADOConnection* per stabilire una connessione con un datastore, invece di fornire semplicemente una stringa di connessione per i componenti di comando e dataset ADO, è che *TADOConnection* consente di controllare molto efficacemente le condizioni e gli attributi della connessione.

Come forzare connessioni asincrone

Utilizzare la proprietà *ConnectOptions* per forzare una connessione asincrona. Le connessioni asincrone permettono a un'applicazione di continuare l'elaborazione senza attendere che la connessione sia aperta completamente.

Per impostazione predefinita, *ConnectionOptions* è impostata a *coConnectUnspecified* che consente al server di decidere il miglior tipo di connessione. Per rendere asincrona la connessione in modo esplicito, impostare *ConnectOptions* a *coAsyncConnect*.

Le routine di esempio riportate di seguito attivano e disattivano le connessioni asincrone nel componente connessione specificato:

```
void __fastcall TForm1::AsyncConnectButtonClick(TObject *Sender)
{
  ADOConnection1->Close();
  ADOConnection1->ConnectOptions = coAsyncConnect;
  ADOConnection1->Open();
}

void __fastcall TForm1::ServerChoiceConnectButtonClick(TObject *Sender)
{
  ADOConnection1->Close();
  ADOConnection1->ConnectOptions = coConnectUnspecified;
  ADOConnection1->Open();
}
```

Controllo dei timeout

Le proprietà *ConnectionTimeout* e *CommandTimeout* consentono di controllare il periodo di tempo che deve trascorrere prima che eventuali tentativi di comando o di connessione siano considerati come non andati a buon fine e quindi annullati.

ConnectionTimeout specifica il periodo di tempo, in secondi, che deve trascorrere prima che un tentativo di connessione al datastore sia considerato come non riuscito. Se la connessione, non viene portata a termine con successo prima della scadenza del periodo di tempo specificato in *ConnectionTimeout*, il tentativo di connessione viene annullato:

```
ADOConnection1->ConnectionTimeout = 10; // seconds
```

```
ADOConnection1->Open();
```

CommandTimeout specifica il periodo di tempo, in secondi, che deve trascorrere prima che un tentativo di comando sia considerato come non riuscito. Se il comando, iniziato da una chiamata al metodo *Execute* non è stato portato a termine con successo prima della scadenza del periodo di tempo specificato in *CommandTimeout*, il comando viene annullato e ADO genera un'eccezione:

```
ADOConnection1->ConnectionTimeout = 10;
ADOConnection1->Execute("DROP TABLE Employee1997", cmdText, TExecuteOptions());
```

Indicazione dei tipi di operazione che la connessione supporta

Le connessioni ADO vengono stabilite utilizzando una modalità specifica, simile a quella utilizzata quando si apre un file. La modalità di connessione determina le autorizzazioni di cui la connessione dispone e perciò i tipi di operazione (come lettura e scrittura) effettuabili.

Utilizzare la proprietà *Mode* per indicare la modalità di connessione. I valori possibili sono elencati nella Tabella 25.2:

Tabella 25.2 Modalità di connessione ADO

Modalità di connessione	Significato
cmUnknown	Le autorizzazioni della connessione non sono ancora impostate o non possono essere determinate.
cmRead	La connessione dispone di autorizzazione per sola lettura.
cmWrite	La connessione dispone di autorizzazione per sola scrittura.
cmReadWrite	La connessione dispone di autorizzazione per lettura e scrittura.
cmShareDenyRead	Impedisce ad altri di stabilire la connessione con autorizzazione di lettura.
cmShareDenyWrite	Impedisce ad altri di stabilire la connessione con autorizzazione di scrittura.
cmShareExclusive	Impedisce ad altri di stabilire la connessione.
cmShareDenyNone	Impedisce ad altri di stabilire la connessione con qualsiasi autorizzazione.

I valori possibili di *Mode* corrispondono ai valori *ConnectModeEnum* della proprietà *Mode* sull'oggetto di connessione ADO sottostante. Per ulteriori informazioni si questi valori, consultare la Guida di Microsoft Data Access SDK.

Come specificare se la connessione avvia in automatico le transazioni

Utilizzare la proprietà *Attributes* per controllare la ritenzione da parte del componente connessione delle modifiche confermate e di quelle annullate. Se il componente connection utilizza la ritenzione delle convalide, ogni volta che l'applicazione convalida una transazione, viene avviata automaticamente una nuova transazione. Se il componente connection utilizza la ritenzione degli annullamenti, ogni volta che l'applicazione annulla la transazione viene avviata automaticamente una nuova transazione.

Attributes è un set che può contenere una, entrambe o nessuna delle costanti *xaCommitRetaining* e *xaAbortRetaining*. Quando *Attributes* contiene *xaCommitRetaining*, la connessione utilizza la ritenzione delle convalide. Quando *Attributes* contiene *xaAbortRetaining*, utilizza la ritenzione degli annullamenti.

Controllare se la ritenzione delle modifiche confermate o di quelle annullate è attiva utilizzando il metodo Set *Contains*. Si attiva la ritenzione delle convalide o degli annullamenti aggiungendo il valore appropriato alla proprietà *Attributes* ; si disattivano sottraendo il valore. Le routine di esempio riportate di seguito attivano e disattivano la ritenzione delle modifiche in un componente ADO connection.

```
void __fastcall TForm1::RetainingCommitsOnButtonClick(TObject *Sender)
{
  ADOConnection1->Close()
  if (!ADOConnection1->Attributes.Contains(xaCommitRetaining))
    ADOConnection1->Attributes = TXactAttributes() << xaCommitRetaining;
  ADOConnection1->Open()
}
void __fastcall TForm1::RetainingCommitsOffButtonClick(TObject *Sender)
{
  ADOConnection1->Close()
  if (ADOConnection1->Attributes.Contains(xaCommitRetaining))
    ADOConnection1->Attributes = TXactAttributes() >> xaCommitRetaining;
  ADOConnection1->Open()
}
```

Accesso ai comandi della connessione

Come altri componenti per la connessione ai database, si accede ai dataset associati alla connessione mediante le proprietà *DataSets* e *DataSetCount*. Tuttavia, *dbGo* include anche oggetti *TADOCommand*, che non sono dataset, ma che hanno una relazione simile con il componente connection.

È possibile utilizzare le proprietà *Commands* e *CommandCount* di *TADOConnection* per accedere agli oggetti di comando ADO associati nello stesso modo in cui si utilizzano le proprietà *DataSets* e *DataSetCount* per accedere ai dataset associati. A differenza di *DataSets* e *DataSetCount*, che elencano solo i dataset attivi, *Commands* e *CommandCount* forniscono riferimenti a tutti i componenti *TADOCommand* associati al componente connessione.

Commands è un array a base zero di riferimenti a componenti comandi ADO. La proprietà *CommandCount* fornisce un conteggio totale di tutti i comandi elencati in *Commands*. Si possono utilizzare queste proprietà insieme per passare in sequenza attraverso tutti i comandi che utilizzano un componente connection, come illustrato nel seguente codice:

```
for (int i = 0; i < ADOConnection2->CommandCount; i++)
  ADOConnection2->Commands[i]->Execute();
```

Eventi di connessione ADO

Oltre agli eventi soliti, che si verificano per tutti i componenti di connessione ai database, *TADOConnection* genera diversi altri eventi che si verificano durante il normale impiego.

Eventi allo stabilirsi di una connessione

Oltre agli eventi *BeforeConnect* e *AfterConnect* che sono comuni a tutti i componenti di connessione ai database, *TADOConnection* quando stabilisce una connessione genera anche gli eventi *OnWillConnect* e *OnConnectComplete*. Questi eventi si verificano dopo *BeforeConnect*.

- *OnWillConnect* si verifica prima che il provider ADO stabilisca una connessione. Consente di fare le ultime modifiche alla stringa di connessione, fornisce lo username e la password se il login viene gestito manualmente, forza una connessione asincrona, oppure annulla la connessione prima che sia stabilita.
- *OnConnectComplete* si verifica dopo l'apertura della connessione. Siccome *TADOConnection* può rappresentare le connessioni asincrone, conviene utilizzare *OnConnectComplete*, che entra in gioco dopo che la connessione è aperta o quando non va a buon fine a causa di una condizione di errore, invece dell'evento *AfterConnect*, che entra in gioco dopo che il componente connection ha incaricato il provider ADO di stabilire la connessione, ma non necessariamente dopo che la connessione è stata aperta.

Eventi alla chiusura del collegamento

Oltre agli eventi *BeforeDisconnect* e *AfterDisconnect*, comuni a tutti i componenti di connessione ai database, *TADOConnection* dopo avere chiuso una connessione genera anche un evento *OnDisconnect*. *OnDisconnect* entra in azione dopo che la connessione è stata chiusa, ma prima che vengano chiusi i dataset associati e prima dell'evento *AfterDisconnect*.

Eventi durante la gestione delle transazioni

Il componente connessione di ADO possiede una serie di eventi che consentono di determinare se sono stati completati i processi correlati alla transazione. Questi eventi indicano al datastore quando un processo di transazione, iniziato da un metodo *BeginTrans*, *CommitTrans* e *RollbackTrans*, è stato portato a termine con successo.

- L'evento *OnBeginTransComplete* si verifica non appena il datastore ha iniziato con successo una transazione, in conseguenza di una chiamata al metodo *BeginTrans*.
- L'evento di *OnCommitTransComplete* si verifica dopo che una transazione è stata confermata con successo, in conseguenza ad una chiamata al metodo *CommitTrans*.
- L'evento *OnRollbackTransComplete* si verifica dopo che una transazione è stata annullata con successo, in conseguenza ad una chiamata al metodo *RollbackTrans*.

Altri eventi

I componenti connection ADO introducono altri due eventi utilizzabili per rispondere alle notifiche provenienti dall'oggetto di connessione ADO sottostante:

- L'evento *OnExecuteComplete* si verifica dopo che il componente connection ha eseguito un comando sul datastore (ad esempio, dopo avere chiamato il metodo *Execute*). *OnExecuteComplete* indica se l'esecuzione ha avuto successo.

- L'evento *OnInfoMessage* entra in azione quando l'oggetto connection sottostante fornisce informazioni dettagliate, dopo il completamento di un'operazione. Il gestore di evento *OnInfoMessage* riceve l'interfaccia verso un oggetto ADO Error contenente le informazioni dettagliate e un codice di stato che segnala se l'operazione è andata a buon fine.

Uso di dataset ADO

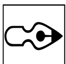
I componenti dataset ADO incapsulano l'oggetto ADO Recordset. Ereditano le funzionalità di dataset comuni, descritte nel Capitolo 22, "I dataset", e le implementano tramite ADO. Per utilizzare un dataset ADO, è necessario prendere familiarità con queste funzioni comuni.

Alle funzioni di dataset comuni, tutti i dataset ADO aggiungono proprietà , eventi e metodi per

- Il collegamento a un datastore ADO.
- L'accesso all'oggetto Recordset sottostante.
- Filtraggio dei record in base a segnalibri.
- Ottenimento di record in modo asincrono.
- Esecuzione aggiornamenti batch (uso degli aggiornamenti in cache).
- Utilizzo di file su disco per memorizzare dati.

Ci sono quattro dataset ADO:

- *TADOTable*, un dataset di tipo tabella che rappresenta tutte le righe e le colonne di una singola tabella di database. Per informazioni sull'utilizzo di *TADOTable* e di altri dataset di tipo tabella, consultare "Utilizzo dei dataset di tipo tabella" a pagina 22-26.
- *TADOQuery*, un dataset di tipo query che incorpora un'istruzione *SQL* e permette alle applicazioni di accedere ai record risultanti, se esistono. Per informazioni sull'utilizzo di dataset di tipo *TADOquery* e di altri di tipo query, vedere "Utilizzo di dataset di tipo query" a pagina 22-43.
- *TADOStoredProc*, un dataset di tipo procedura registrata, che esegue una procedura registrata definita su un server di database. Per informazioni sull'utilizzo di *TADOStoredProc* e di altri dataset di tipo procedura registrata, consultare "Utilizzo di dataset di tipo procedura registrata" a pagina 22-52.
- *TADODataSet*, un dataset generico che incorpora le funzionalità degli altri tre tipi. Per una descrizione delle funzioni particolari di *TADODataSet*, consultare "Utilizzo di TADODataSet" a pagina 25-16.



Quando ADO viene utilizzato per accedere a informazioni di database, non è necessario utilizzare un dataset come *TADOQuery* per rappresentare comandi *SQL* che non restituiscono un cursor. Invece, è possibile utilizzare *TADOCommand*, un componente semplice che non è un dataset. Per dettagli su *TADOCommand*, consultare "Utilizzo di oggetti Command" a pagina 25-17.

Connessione di un dataset ADO a un datastore

I dataset ADO possono essere connessi a un datastore ADO in modo collettivo o singolarmente.

Se si vogliono connettere i datasets in modo collettivo, impostare la proprietà *Connection* di ciascun componente dataset a un componente *TADOConnection*. Ogni dataset utilizza quindi la connessione del componente connection ADO.

```
ADODataSet1->Connection = ADOConnection1;
ADODataSet2->Connection = ADOConnection1;
...
```

La connessione collettiva di componenti datasets presenta, tra gli altri, i seguenti vantaggi:

- I componenti datasets condividono gli attributi dell'oggetto connessione.
- Deve essere impostata solo una connessione: quella di *TADOConnection*.
- I dataset possono partecipare alle transazioni.

Per ulteriori informazioni sull'impiego di *TADOConnection*, consultare "Connessione a datastore ADO" a pagina 25-2.

Se si vogliono connettere singolarmente i datasets, impostare la proprietà *ConnectionString* di ciascun dataset.Ogni dataset che utilizza *ConnectionString* stabilisce la propria connessione al datastore, in modo totalmente indipendente da qualsiasi altra connessione al dataset nell'applicazione.

La proprietà *ConnectionString* dei dataset ADO funziona come la proprietà *ConnectionString* di *TADOConnection*: è un set di parametri di connessione delimitati da punti e virgola come i seguenti:

```
ADODataSet1->ConnectionString = "Provider=YourProvider;Password=SecretWord;";
ADODataSet1->ConnectionString += "User ID=JaneDoe;SERVER=PURGATORY";
ADODataSet1->ConnectionString += "UID=JaneDoe;PWD=SecretWord;"
ADODataSet1->ConnectionString += "Initial Catalog=Employee";
```

In progettazione, è possibile utilizzare il *Connection* String Editor che aiuta a costruire la stringa di connessione. Per informazioni sulle stringhe di connessione, consultare "Connessione a un datastore mediante TADOConnection" a pagina 25-3.

Operazioni con set di record

La proprietà *RecordsetState* consente di accedere in modo diretto all'oggetto recordset ADO sottostante al componente dataset. Utilizzando questo oggetto, è possibile da un'applicazione accedere alle proprietà e chiamare i metodi dell'oggetto recordset. L'uso di *Recordset* per accedere direttamente all'oggetto recordset ADO sottostante richiede una buona conoscenza operativa degli oggetti ADO in generale e dell'oggetto recordset ADO in particolare. Non si consiglia l'utilizzo diretto dell'oggetto recordset a meno che non si abbia familiarità con le operazioni sugli oggetti recordset. Per informazioni specifiche sull'utilizzo di oggetti recordset ADO, consultare la Guida di Microsoft Data Access SDK.

La proprietà *RecordsetState* indica lo stato attuale dell'oggetto recordset sottostante. RecordSetState corrisponde alla proprietà *State* dell'oggetto recordset ADO. Il valore di *RecordsetState* è *stOpen*, *stExecuting*, oppure *stFetching*. (*TObjectState*, il tipo della

proprietà *RecordsetState*, definisce altri valori, ma solo *stOpen*, *stExecuting*, e *stFetching* riguardano i recordset). Un valore pari a *stOpen* indica che il recordset è attivo. Un valore pari a *stExecuting* indica che sta eseguendo un comando. Un valore pari a *stFetching* indica che sta recuperando righe dalla tabella o dalle tabelle associate).

Utilizzare i valori di *RecordsetState* per compiere azioni dipendenti dallo stato attuale del dataset. Per esempio, una routine di aggiornamento dei dati potrebbe controllare la proprietà *RecordsetState* per verificare se il dataset è attivo e non sta eseguendo altre attività come una connessione o il reperimento di dati.

Filtraggio di record mediante segnalibri

I dataset ADO dispongono della funzionalità , comune ai dataset, di contrassegnare e restituire record specifici sfruttando i segnalibri. Inoltre, come altri dataset, impiegano i filtri per limitare i record disponibili nel dataset. I dataset ADO dispongono di un'ulteriore funzione che combina le due funzionalità comuni dei dataset: la capacità di filtrare un insieme di record identificati da segnalibri.

Per filtrare in base a una serie di segnalibri:

1. Mediante il metodo *Bookmark*, contrassegnare i record da includere nel dataset filtrato.
2. Mediante il metodo *FilterOnBookmarks*, filtrare il dataset in modo che vengano visualizzati solo i record contrassegnati.

La procedura è illustrata qui di seguito:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  TBookmarkStr BM1;
  TBookmarkStr BM2;
  BM1 = ADODataSet1->Bookmark;
  BMList->Add(BM1);
  ADODataSet1->MoveBy(3);
  BM2 = ADODataSet1->Bookmark;
  BMList->Add(BM2);
  ADODataSet1->FilterOnBookmarks(ARRAYOFCONST((BM1,BM2)));
}
```

Tenere presente che, nell'esempio precedente, i segnalibri vengono inseriti in un oggetto lista chiamato BMList. Questo perché l'applicazione possa liberare in seguito i segnalibri quando non servono più.

Per dettagli sull'impiego dei segnalibri, consultare "Contrassegno e ritorno ai record" a pagina 22-9. Per dettagli su altri tipi di filtri, consultare "Visualizzazione e modifica di un sottoinsieme di dati usando i filtri" a pagina 22-13.

Prelievo di record in modalità asincrona

A differenza di altri dataset, i dataset ADO possono prelevare i dati in modo asincrono. Questo permette all'applicazione di fare altro, mentre il dataset si riempie di dati presi dal datastore.

Per controllare se il dataset preleva i dati in modo asincrono, se pure li preleva, si utilizza la proprietà *ExecuteOptions*. *ExecuteOptions* regola il modo in cui il dataset

preleva i record quando si chiama *Open* o si imposta la proprietà *Active* a **true**. Se il dataset rappresenta una query o una procedura registrata che non restituisce record, *ExecuteOptions* regola il modo in cui vengono eseguite la query o la procedura registrata quando si chiamano *ExecSQL* o *ExecProc*.

ExecuteOptions è un insieme che include zero o più dei seguenti valori:

Tabella 25.3 Opzioni di esecuzione dei dataset ADO

Opzione di esecuzione	Significato
eoAsyncExecute	Il comando o l'operazione di prelievo dati vengono eseguiti in modo asincrono.
eoAsyncFetch	Il dataset prima preleva in modo sincrono il numero di record specificati dalla proprietà *CacheSize*, quindi preleva le eventuali righe rimanenti in modo asincrono.
eoAsyncFetchNonBlocking	I prelievi asincroni di dati o l'esecuzione di comandi non interrompono il thread di esecuzione attivo.
eoExecuteNoRecords	Un comando o una procedura registrata che non restituiscono dati. Le eventuali righe recuperate, vengono eliminate e non restituite.

Uso di aggiornamenti batch

Un approccio per la gestione degli aggiornamenti in cache consiste nel connettere il dataset ADO a un dataset client utilizzando un provider di dataset. Questo approccio è discusso in "Utilizzo di un dataset client per registrare in cache gli aggiornamenti" a pagina 27-17.

Tuttavia, i componenti dataset ADO sono in grado di supportare gli aggiornamenti memorizzati nella cache, che chiamano aggiornamenti batch. La seguente tabella elenca le corrispondenze tra la memorizzazione di aggiornamenti nella cache utilizzando un dataset client, oppure utilizzando le funzionalità di aggiornamento batch:

Tabella 25.4 Confronto tra gli aggiornamenti in cache di ADO e gli aggiornamenti in cache nel dataset client

Dataset ADO	TClientDataSet	Descrizione
LockType	Non utilizzato: i dataset client memorizzano sempre gli aggiornamenti nella cache	Specifica se il dataset è aperto in modo aggiornamento batch.
CursorType	Non utilizzato: i dataset client funzionano sempre con un'"istantanea" in memoria dei dati	Specifica in che misura il dataset ADO è isolato dalle modifiche sul server.
RecordStatus	UpdateStatus	Indica l'eventuale tipo di aggiornamento che si è verificato sulla riga corrente. *RecordStatus* fornisce più informazioni di *UpdateStatus*.
FilterGroup	StatusFilter	Specifica quali tipi di record sono disponibili. *FilterGroup* fornisce una gamma più ampia di informazioni.

Tabella 25.4 Confronto tra gli aggiornamenti in cache di ADO e gli aggiornamenti in cache nel dataset client

Dataset ADO	TClientDataSet	Descrizione
UpdateBatch	ApplyUpdates	Applica al server di database gli aggiornamenti nella cache.A differenza di *ApplyUpdates*, *UpdateBatch* permette di limitare i tipi di aggiornamenti da applicare.
CancelBatch	CancelUpdates	Elimina gli aggiornamenti in sospeso e torna ai valori originari. A differenza di *CancelUpdates*, *CancelBatch* permette di limitare i tipi di aggiornamenti da annullare.

L'utilizzo delle funzionalità degli aggiornamenti batch dei componenti dataset ADO può essere schematizzata come segue:

- Apertura del dataset in modalità aggiornamento batch
- Ispezione dello stato di aggiornamento delle singole righe
- Filtraggio di più righe in base allo stato di aggiornamento
- Applicazione degli aggiornamenti batch alle tabelle di base
- Annullamento degli aggiornamenti batch

Apertura del dataset in modalità aggiornamento batch

Per aprire un dataset ADO in modalità aggiornamento batch, devono essere soddisfatte le seguenti condizioni:

1. La proprietà *CursorType* del componente deve essere impostata a *ctKeySet* (il valore predefinito della proprietà) oppure a *ctStatic*.
2. La proprietà *LockType* deve essere impostata a *ltBatchOptimistic*.
3. Il comando deve essere una query SELECT.

Prima di attivare il componente dataset, impostare le proprietà *CursorType* e *LockType* ai valori indicati in precedenza. Assegnare un'istruzione SELECT alla proprietà *CommandText* del componente (per un *TADODataSet*) oppure alla proprietà *SQL* (nel caso di un *TADOQuery*). Per i componenti *TADOStoredProc*, impostare *ProcedureName* al nome di una procedura registrata che restituisce un set risultato. Queste proprietà possono essere impostate in fase di progetto mediante l'Object Inspector oppure in fase di esecuzione dal programma. L'esempio seguente mostra la preparazione di un componente *TADODataSet* per la modalità aggiornamento batch.

```
ADODataSet1->CursorLocation = clUseClient;
ADODataSet1->CursorType = ctStatic;
ADODataSet1->LockType = ltBatchOptimistic;
ADODataSet1->CommandType = cmdText;
ADODataSet1->CommandText = "SELECT * FROM Employee";
```

Dopo aver aperto un dataset in modalità aggiornamento batch, tutte le modifiche ai dati vengono inserite in cache invece di essere applicate direttamente alle tabelle di base.

Ispezione dello stato di aggiornamento delle singole righe

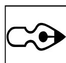
Lo stato di aggiornamento di una certa riga si determina rendendola attiva e quindi ispezionando la proprietà *RecordStatus* del componente dati ADO. *RecordStatus* rispecchia lo stato di aggiornamento della riga corrente e solamente di quella riga.

```
switch (ADOQuery->RecordStatus)
{
  case rsUnmodified:
    StatusBar1->Panels->Items[0]->Text = "Unchanged record";
    break;
  case rsModified:
    StatusBar1->Panels->Items[0]->Text = "Changed record";
    break;
  case rsDeleted:
    StatusBar1->Panels->Items[0]->Text = "Deleted record";
    break;
  case rsNew:
    StatusBar1->Panels->Items[0]->Text = "New record";
    break;
}
```

Filtraggio di più righe in base allo stato di aggiornamento

È possibile filtrare un recordset per visualizzare solo quelle righe che appartengono a un gruppo di righe con lo stesso stato di aggiornamento utilizzando la proprietà *FilterGroup*. Impostare *FilterGroup* alla costante *TFilterGroup* che rappresenta lo stato di aggiornamento delle righe che si vogliono visualizzare. Un valore pari a *fgNone* (il valore predefinito per questa proprietà) specifica che non è stato applicato alcun filtrazione e tutte le righe (tranne quelle marcate per cancellazione) sono visibili, indipendentemente dal loro stato di aggiornamento. L'esempio seguente provoca la visualizzazione delle sole righe per cui è pendente un aggiornamento batch.

```
FilterGroup = fgPendingRecords;
Filtered = true;
```



Affinché la proprietà *FilterGroup* abbia effetto, la proprietà *Filtered* del componente dataset ADO deve essere impostata a **true**.

Applicazione degli aggiornamenti batch alle tabelle di base

Applicare le modifiche in sospeso ai dati, che non sono state ancora applicate o che sono state annullate, chiamando il metodo *UpdateBatch*. I cambiamenti apportati alle righe che sono state modificate e applicate vengono trasferiti nelle tabelle di base su cui è basato il recordset. Una riga inserita in cache e contrassegnata per la cancellazione provoca la cancellazione della riga corrispondente dalla tabella di base. Un inserimento di record (presente nella cache ma non nella tabella di base) viene aggiunto alla tabella di base. Le righe modificate fanno sì che le colonne nelle righe corrispondenti nelle tabelle di base vengano modificate in base ai nuovi valori di colonna presenti nella cache.

UpdateBatch accetta un parametro, un valore *TAffectRecords*. Facoltativamente, è possibile passare come parametro ad *UpdateBatch* un valore di *TAffectRecords*. L'esempio seguente provoca l'applicazione delle modifiche apportate alla sola riga corrente:

```
ADODataSet1->UpdateBatch(arCurrent);
```

Annullamento degli aggiornamenti batch

Annullare le modifiche ai dati in sospeso, che non sono state ancora annullate o applicate, chiamando il metodo *CancelBatch*. Quando si annullano degli aggiornamenti batch in sospeso, i valori di campo nelle righe sottoposte a modifica che sono state modificate tornano ai valori in vigore prima dell'ultima chiamata a *CancelBatch* o *UpdateBatch*, se l'uno o l'altro è stato chiamato, o prima del lotto corrente di modifiche in sospeso.

CancelBatch accetta un parametro, un valore *TAffectRecords*. Il passaggio di un qualsiasi altro valore, ad eccezione di *arAll*, provoca l'annullamento solo di un sottoinsieme delle modifiche pendenti. L'esempio seguente annulla tutte le modifiche pendenti:

```
ADODataSet1->CancelBatch(arAll);
```

Caricamento e salvataggio di dati su file

I dati reperiti mediante un componente dataset ADO possono essere salvati in un file, sullo stesso computer o su uno differente, in modo da poter eseguire ricerche in una fase successiva. I dati vengono salvati in uno dei due formati proprietari: ADTG e XML. Questi due formati di file sono gli unici formati supportati da ADO. Comunque, non è detto che entrambi i formati siano supportati in tutte le versioni ADO. Consultare la documentazione ADO relativa alla versione utilizzata per determinare quali sono i formati file di salvataggio supportati.

Il salvataggio di dati in un file avviene utilizzando il metodo *SaveToFile*. *SaveToFile* accetta due parametri, il nome del file su cui i dati sono salvati e il formato (ADTG o XML) in cui salvarli. Si indica il formato del file da salvare impostando il parametro *Format* a *pfADTG* o a *pfXML*. Se il file specificato dal parametro *FileName* esiste già , *SaveToFile* solleva una *EOleException*.

Il reperimento di dati da un file avviene utilizzando il metodo *LoadFromFile*. *LoadFromFile* accetta un solo parametro, il nome del file da caricare. Se il file specificato non esiste, *LoadFromFile* solleva un'eccezione *EOleException*. Il componente dataset viene attivato automaticamente appena si esegue la chiamata al metodo *LoadFromFile*.

Nell'esempio seguente, la prima procedura salva il dataset, reperito dal componente *ADODataSet1* di tipo *TADODataSet*, in un file. Il file di destinazione è un file di tipo ADTG di nome SaveFile, salvato su un'unità locale. La seconda procedura carica questo file salvato nel componente *ADODataSet2* di tipo *TADODataSet*.

```
void __fastcall TForm1::SaveBtnClick(TObject *Sender)
{
  if (FileExists("c:\\SaveFile"))
  {
    DeleteFile("c:\\SaveFile");
    Statusbar1->Panels->Items[0]->Text = "Save file deleted!";
  }
  ADODataSet1->SaveToFile("c:\\SaveFile");
}
```

```
void __fastcall TForm1::LoadBtnClick(TObject *Sender)
{
  if (FileExists("c:\\SaveFile"))
    ADODataSet1->LoadFromFile("c:\\SaveFile");
  else
    Statusbar1->Panels->Items[0]->Text = "Save file does not exist!";
}
```

I dataset che effettuano il salvataggio e il caricamento dei dati non devono necessariamente essere nel formato visto in precedenza, nella stessa applicazione o sullo stesso computer. Questo consente un trasferimento dei dati di tipo briefcase da un computer a un altro.

Utilizzo di TADODataSet

TADODataSet è un dataset di uso comune usato per lavorare con i dati di un datastore ADO. A differenza degli altri componenti dataset ADO, *TADODataSet* non è di tipo tabella, di tipo query o di tipo procedura registrata. Tuttavia, funziona indifferentemente come uno qualsiasi di questi tre tipi:

- Come un dataset di tipo tabella, *TADODataSet* permette di rappresentare tutte le righe e le colonne di una singola tabella di database. Per utilizzarlo in questo modo, si imposta la proprietà *CommandType* a *cmdTable* e la proprietà *CommandText* al nome della tabella. *TADODataSet* supporta attività di tipo tabellare come
  - Assegnazione di indici per ordinare i record o per costituire la base di ricerche basate su record. Oltre che con le proprietà e i metodi di indicizzazione standard descritti in "Ordinamento dei record con indici" a pagina 22-27, *TADODataSet* consente, impostando la proprietà *Sort*, di ordinare utilizzando indici temporanei. Le ricerche basate su indice svolte tramite il metodo *Seek* utilizzano l'indice corrente.
  - Svuotamento del dataset. Il metodo *DeleteRecords* consente un controllo più incisivo che non i metodi analoghi di altri dataset di tipo tabella, perché permette di specificare quali record cancellare.

  Le attività di tipo tabella supportate da *TADODataSet* sono disponibili anche quando non si sta utilizzando un *CommandType* di *cmdTable*.
- Come i dataset di tipo query, *TADODataSet* permette di specificare un singolo comando *SQL* da eseguire quando il dataset viene aperto. Per utilizzarlo in questo modo, impostare la proprietà *CommandType* a *cmdText* e la proprietà *CommandText* al comando *SQL* da eseguire. In progettazione, è possibile fare doppio clic sulla proprietà *CommandText* nell'Object Inspector per farsi aiutare dal Command Text editor a impostare il il comando *SQL*. *TADODataSet* supporta attività query-type come
  - Utilizzo dei parametri nel testo di query. Per i dettagli sulle query parametrizzate consultare "Utilizzo dei parametri nelle query" a pagina 22-46.

  - Configurare le relazioni master/detail utilizzando parametri. Per i dettagli di impiego, consultare "Impostazione di relazioni master/detail mediante parametri" a pagina 22-49.
  - Preparazione in anticipo della query per migliorare le prestazioni impostando a **true** la proprietà *Prepared*.

- Come un dataset di tipo procedura registrata, *TADODataSet* permette di specificare una procedura registrata che è eseguita quando si apre il dataset. Per utilizzarlo in questo modo, si impostano a *cmdStoredProc* la proprietà *CommandType*, e al nome della procedura registrata la proprietà *CommandText*. *TADODataSet* supporta le attività di tipo procedura memorizzate come
  - Operazioni con i parametri di una procedura registrata. Per i dettagli sui parametri di una procedura registrata consultare "Operazioni con i parametri delle procedure registrate" a pagina 22-53.
  - Prelievo di più set risultato. Consultare "Prelievo di più set risultato" a pagina 22-57.
  - Preparazione in anticipo della procedura registrata per migliorare le prestazioni impostando la proprietà *Prepared* a **true**.

Inoltre, *TADODataSet* permette di gestire dati memorizzati nei file impostando la proprietà *CommandType* a *cmdFile* e la proprietà *CommandText* al nome al file.

Prima di impostare le proprietà *CommandText* e *CommandType*, bisogna connettere *TADODataSet* con un datastore impostando la proprietà *Connection* o *ConnectionString*. La procedura è spiegata in "Connessione di un dataset ADO a un datastore" a pagina 25-10. In alternativa, si può utilizzare un oggetto DataSpace RDS per connettere *TADODataSet* a un server applicativo basato su ADO. Per utilizzare un oggetto DataSpace RDS, impostare la proprietà *RDSConnection* a un oggetto *TRDSConnection*.

Utilizzo di oggetti Command

Nell'ambiente ADO, i comandi sono rappresentazioni testuali di richieste di azioni specifiche per il provider. Tipicamente, sono istruzioni SQL di tipo DDL (Data Definition Language) o DML (Data Manipulation Language). Il linguaggio utilizzato nei comandi è specifico per ogni provider, ma di solito conforme con lo standard *SQL*-92 del linguaggio SQL.

Benché si possano sempre eseguire i comandi mediante *TADOQuery*, non conviene sobbarcarsi il carico di elaborazione causato dall'impiego di un componente dataset, specialmente se il comando non restituisce un set risultato. Come alternativa è possibile utilizzare il componente *TADOCommand* che è un oggetto leggero progettato per eseguire comandi, uno alla volta. *TADOCommand* è destinato principalmente all'esecuzione di quei comandi che non restituiscono un set risultato, come le istruzioni *SQL* di tipo DDL (Data Definition Language). Tuttavia, grazie a una ridefinizione del metodo *Execute*, è in grado di restituire un set risultato che può essere assegnato alla proprietà *RecordSet* di un componente ADO dataset.

Di solito, lavorare con *TADOCommand* è molto simile a lavorare con *TADODataSet*, salvo per il fatto che non si possono utilizzare i metodi di dataset standard per prelevare dati, spostarsi tra i record, modificare i dati e così via. Gli oggetti *TADOCommand* si connettono a un datastore proprio come i dataset ADO. Per i dettagli, consultare "Connessione di un dataset ADO a un datastore" a pagina 25-10 for details.

Le sezioni seguenti spiegano come specificare ed eseguire i comandi utilizzando *TADOCommand*.

Specifica del comando

I comandi di un componente *TADOCommand* si specificano mediante la proprietà *CommandText*. Come *TADODataSet*, *TADOCommand* permette di specificare il comando in vari modi, a seconda della proprietà *CommandType*. I possibili valori di *CommandType* includono: *cmdText* (usato se il comando è un'istruzione *SQL*), *cmdTable* (se è un nome di tabella) e *cmdStoredProc* (se il comando è il nome di una procedura registrata). In fase di progettazione, selezionare dall'elenco nell'Object Inspector il tipo di comando appropriato. In esecuzione, assegnare alla proprietà *CommandType* un valore di tipo *TCommandType*.

```
ADOCommand1->CommandText = "AddEmployee";
ADOCommand1->CommandType = cmdStoredProc;
...
```

Se non viene specificato alcun tipo particolare, si lascia al server l'incombenza di decidere al meglio in base al comando contenuto in *CommandText*.

CommandText può contenere il testo di una query *SQL* che include parametri, o il nome di una procedura registrata che utilizza parametri. È necessario fornire i valori dei parametri, che vengono collegati ai parametri prima di eseguire il comando. Per i dettagli, consultare "Gestione dei parametri di comando" a pagina 25-20.

Utilizzo del metodo Execute

Prima che il componente *TADOCommand* possa eseguire il suo comando, deve aver stabilito una connessione valida con un datastore. Il collegamento viene stabilito esattamente come avviene con i dataset ADO. Per i dettagli, consultare "Connessione di un dataset ADO a un datastore" a pagina 25-10.

Per eseguire il comando, chiamare il metodo *Execute*. *Execute* è un metodo caricato in sovrapposizione che permette di scegliere il modo più appropriato per eseguire il comando.

Per i comandi che non richiedono parametri e dei quali non interessa il numero di record interessati, si chiama *Execute* senza nessun parametro:

```
ADOCommand1->CommandText = "UpdateInventory";
ADOCommand1->CommandType = cmdStoredProc;
ADOCommand1->Execute();
```

Altre versioni di *Execute* permettono di fornire i valori dei parametri utilizzando un array Variant, e ottenendo così il numero di record interessati dal comando.

Per informazioni sull'esecuzione di comandi che restituiscono un set risultato, consultare "Reperimento di set risultato con comandi" a pagina 25-19.

Annullamento di comandi

Se si esegue il comando in modo asincrono, allora dopo avere chiamato *Execute* è possibile interrompere l'esecuzione chiamando il metodo *Cancel*:

```
void __fastcall TDataForm::ExecuteButtonClick(TObject *Sender)
{
  ADOCommand1->Execute();
}
void __fastcall TDataForm::CancelButtonClick(TObject *Sender)
{
  ADOCommand1->Cancel();
}
```

Il metodo *Cancel* sortisce un effetto solo se c'è un comando in sospeso e il comando era stato eseguito in modo asincrono (il parametro *ExecuteOptions* del metodo *Execute* contiene *eoAsynchExecute*). Un comando viene definito "in sospeso" se è stato chiamato il metodo *Execute* ma il comando non è stato ancora completato oppure è andato in time-out.

Se un comando non viene annullato o completato entro il numero di secondi specificato nella proprietà *CommandTimeout*, il comando va in time-out. Per impostazione predefinita, i comandi vanno in time-out dopo 30 secondi.

Reperimento di set risultato con comandi

A differenza dei componenti *TADOQuery*, che utilizzano vari metodi di esecuzione, a seconda che restituiscano un set risultato o meno, *TADOCommand* utilizza sempre il comando *Execute* per eseguire il comando, che restituisca o no un set risultato. Quando restituisce un set risultato, *Execute* restituisce un'interfaccia verso l'interfaccia ADO _RecordSet.

Il modo più conveniente per lavorare con questa interfaccia è di assegnarla alla proprietà *RecordSet* di un dataset ADO.

Ad esempio, il seguente codice utilizza *TADOCommand* (*ADOCommand1*) per eseguire una query SELECT, che restituisce un set risultato. Questo set risultato viene poi assegnato alla proprietà *RecordSet* di un componente *TADODataSet* (*ADODataSet1*).

```
ADOCommand1->CommandText = "SELECT Company, State ";
ADOCommand1->CommandText += "FROM customer ";
ADOCommand1->CommandText += "WHERE State = :StateParam";
ADOCommand1->CommandType = cmdText;
ADOCommand1->Parameters->ParamByName("StateParam")->Value = "HI";
ADOCommand1->Recordset = ADOCommand1->Execute();
```

Non appena il set risultato viene assegnato alla proprietà *Recordset* del componente dataset ADO, il componente dataset viene automaticamente attivato e i dati diventano disponibili.

Gestione dei parametri di comando

Sono due i modi in cui un oggetto *TADOCommand* può utilizzare i parametri:

- La proprietà *CommandText* può specificare una query che include dei parametri. Lavorare con le query parametrizzate in *TADOCommand* è come usare una query parametrizzata in un dataset ADO. Per i dettagli sulle query parametrizzate consultare "Utilizzo dei parametri nelle query" a pagina 22-46.
- La proprietà *CommandText* può specificare una procedura registrata che utilizza parametri. I parametri di una procedura registrata funzionano più o meno come *TADOCommand* con i dataset ADO. Per i dettagli sui parametri di una procedura registrata consultare "Operazioni con i parametri delle procedure registrate" a pagina 22-53.

Ci sono due modi per fornire valori dei parametri quando si lavora con *TADOCommand*: si possono fornire quando si chiama il metodo *Execute*, oppure specificarli anticipatamente mediante la proprietà *Parameters*.

Il metodo *Execute* è caricato in sovrapposizione per includere versioni che accettano un set di valori di parametro sotto forma di array Variant. La cosa è utile quando si desidera fornire velocemente i valori dei parametri senza l'aggravio di configurare la proprietà *Parameters*:

```
Variant Values[2];
Values[0] = Edit1->Text;
Values[1] = Date();
ADOCommand1.Execute(VarArrayOf(Values,1));
```

Quando si lavora con le procedure registrate che restituiscono parametri di output, bisogna invece utilizzare la proprietà *Parameters*. Anche se non si ha bisogno di leggere i parametri di output, conviene usare la proprietà *Parameters*, che permette di fornire i parametri in fase di progetto e di lavorare con le proprietà *TADOCommand* nello stesso modo in cui si lavora con i parametri sui dataset.

Quando si imposta la proprietà *CommandText*, la proprietà *Parameters* viene aggiornata automaticamente per riflettere i parametri nella query, o quelli utilizzati dalla procedura registrata. In progettazione, accedere ai parametri utilizzando il Parameter Editor, attivato facendo clic nell'Object Inspector sul pulsante ellissi della proprietà *Parameters*. In fase di esecuzione, per impostare (o recuperare) i valori di ciascun parametro utilizzare le proprietà e i metodi di *TParameter*.

```
ADOCommand1->CommandText = "INSERT INTO Talley ";
ADOCommand1->CommandText += "(Counter) ";
ADOCommand1->CommandText += "VALUES (:NewValueParam)";
ADOCommand1->CommandType = cmdText;
ADOCommand1->Parameters->ParamByName("NewValueParam")->Value = 57;
ADOCommand1->Execute()
```

Capitolo

26

Uso di dataset unidirezionali

dbExpress è un gruppo di driver di database leggeri che consentono un accesso rapido ai server di database SQL. Per ogni database supportato, *dbExpress* dispone di un driver che adatta il software specifico del server a un set di classi uniformi di *dbExpress*. Quando si distribuisce un'applicazione database che utilizza *dbExpress*, con i file dell'applicazione generata è necessario includere solamente una dll (il driver specifico per il server.

dbExpress consente di accedere a database utilizzando dataset unidirezionali. I dataset unidirezionali sono progettati per consentire un rapido accesso alle informazioni di database, con un minimo appesantimento. Come altri dataset, sono in grado di inviare un comando SQL al server di database e, se il comando restituisce un set di record, ottengono un cursor per accedere a quei record. Tuttavia, i dataset unidirezionali possono recuperare solo un cursor unidirezionale. Essi non memorizzano i dati in memoria, il che li rende più rapidi e bisognosi di meno risorse rispetto ad altri tipi di dataset. Tuttavia, poiché i record non vengono bufferizzati, i dataset unidirezionali sono anche meno flessibili di altri dataset. Molte funzionalità introdotte da *TDataSet* o non sono implementate nei dataset unidirezionali o causano il sollevamento di eccezioni. Per esempio:

- Gli unici metodi di navigazione supportati sono i metodi *First* e *Next*. La maggior parte degli altri metodi solleva eccezioni. Alcuni, come i metodi coinvolti nel supporto dei segnalibri, semplicemente non fanno niente.
- Non forniscono alcun supporto interno per l'editing perché tale operazione richiede un buffer per conservare le modifiche. La proprietà *CanModify* è sempre **false** e, pertanto, il tentativo di porre il dataset in modalità editing non va mai a buon fine. È possibile, comunque, utilizzare i dataset unidirezionali per aggiornare i dati utilizzando un comando SQL UPDATE o fornire il tradizionale supporto per l'editing utilizzando un dataset client *dbExpress* o connettendo il dataset a un dataset client (consultare "Collegamento a un altro dataset" a pagina 18-10).
- Non forniscono alcun supporto per i filtri in quanto i filtri operano con più record, il che richiede l'utilizzo di un buffer. Se si prova a filtrare un dataset

unidirezionale, viene sollevata un'eccezione. Tutti i limiti, in base a cui vengono visualizzati i dati, devono essere invece imposti utilizzando il comando SQL che definisce i dati del dataset.

- Non forniscono alcun supporto per i campi di consultazione, cosa che richiede anch'essa un buffer per conservare più record contenenti i valori da consultare. Se si definisce un campo di consultazione su un dataset unidirezionale, non potrà operare correttamente.

Malgrado queste limitazioni, i dataset unidirezionali rappresentano un modo potente per accedere ai dati. Risultano essere un meccanimo di accesso ai dati rapidissimo e molto semplice da utilizzare e distribuire.

Tipi di dataset unidirezionali

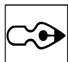
La pagina *dbExpress* della Component palette contiene quattro tipi di dataset unidirezionali: *TSQLDataSet*, *TSQLQuery*, *TSQLTable* e *TSQLStoredProc*.

TSQLDataSet è il più generico dei quattro. È possibile utilizzare un dataset SQL per rappresentare tutti i dati disponibili tramite *dbExpress* o per inviare comandi a un database a cui si accede mediante *dbExpress*. Questo è il componente che si consiglia di utilizzare per operare con tabelle di database nelle nuove applicazioni database.

TSQLQuery è un dataset di tipo query che incapsula un'istruzione SQL e permette alle applicazioni di accedere ai record risultanti, nel caso ve ne siano. Per informazioni sull'utilizzo di dataset di tipo query, vedere "Utilizzo di dataset di tipo query" a pagina 22-43.

TSQLTable è un dataset di tipo tabella che rappresenta tutte le righe e colonne di una singola tabella di database. Per informazioni sull'utilizzo di dataset di tipo tabella, vedere "Utilizzo dei dataset di tipo tabella" a pagina 22-26.

TSQLStoredProc è un dataset di tipo procedura registrata che esegue una procedura registrata definita su un server di database. Per informazioni sull'utilizzo di dataset di tipo procedura registrata, vedere "Utilizzo di dataset di tipo procedura registrata" a pagina 22-52.



La pagina *dbExpress* include anche *TSQLClientDataSet*, che non è un dataset unidirezionale. Piuttosto, è un dataset client che utilizza un dataset unidirezionale internamente per accedere ai suoi dati.

Connessione al server di database

Quando si opera con un dataset unidirezionale la prima operazione consiste nel connetterlo a un server di database. In progettazione, una volta che un dataset ha una connessione attiva a un server di database, l'Object Inspector è in grado di presentare delle liste a discesa con i valori di altre proprietà. Ad esempio, quando si rappresenta una procedura registrata, è necessario avere una connessione attiva prima che l'Object Inspector sia in grado di elencare quali procedure registrate sono disponibili sul server.

La connessione a un server di database è rappresentata da un componente separato *TSQLConnection*. Si opera con *TSQLConnection* come con qualsiasi altro componente di connessione ai database. Per informazioni sull'utilizzo di componenti database connection, vedere il Capitolo 21, "Connessione ai database".

Per utilizzare *TSQLConnection* per connettersi un dataset unidirezionale a un server di database, impostare la proprietà *SQLConnection*. In progettazione, è possibile scegliere il componente SQL connection da una lista a discesa nell'Object Inspector. Se si fa questa assegnazione in esecuzione, è bene accertarsi che la connessione sia attiva:

```
SQLDataSet1->SQLConnection = SQLConnection1;
SQLConnection1->Connected = true;
```

Di solito, tutti i dataset unidirezionali in un'applicazione condividono lo stesso componente connection, a meno che non si stia operando con dati provenienti da più server di database. Tuttavia, potrebbe essere necessario utilizzare un componente connection diverso per ogni dataset nel caso il server non supporti più istruzioni per ogni connessione. Controllare se il server il database richiede una diversa connessione per ogni dataset leggendo la proprietà *MaxStmtsPerConn*. Per impostazione predefinita, *TSQLConnection* genera connessioni appena occorre nel caso il server limiti il numero di istruzioni che possono essere eseguite per ogni singola connessione. Se si desidera tenere una traccia rigorosa delle connessioni utilizzate, impostare la proprietà *AutoClone* a **false**.

Prima di assegnare la proprietà *SQLConnection*, è necessario impostare il componente *TSQLConnection* in modo che identifichi il server di database e tuut i parametri di connessione necessari (inclusi il database da utilizzare sul server, il nome host della macchina su cui il server è in esecuzione, il nome utente, la password, e così via).

Impostazione di TSQLConnection

Per descrivere una connessione al database in modo sufficientemente dettagliato affinché *TSQLConnection* sia in grado di aprire una connessione, è necessario identificare sia il driver da utilizzare sia un insieme di parametri di connessione che vengono passati a quel driver.

Identificazione del driver

Il driver è identificato dalla proprietà *DriverName* che è il nome di un driver installato di *dbExpress*, come INTERBASE, ORACLE, MYSQL o DB2. Il nome del driver è associato a due file

- Il driver *dbExpress*. Questi è una libreria a collegamento dinamico con un nome simile a dbexpint.dll, dbexpora.dll, dbexpmys.dll o dbexpdb2.dll.
- La libreria a collegamento dinamico fornita dal produttore di database per il supporto sul lato client.

La relazione tra questi due file oggetto condivisi e il nome del database viene memorizzata in un file di nome dbxdrivers.ini che viene aggiornato quando si installa un driver *dbExpress*. Di solito, non è necessario occuparsi di questi file perché il componente connection SQL li ricercherà in dbxdrivers.ini non appena si imposterà

il valore della proprietà *DriverName*. Quando si imposta la proprietà *DriverName*, *TSQLConnection* imposta automaticamente le proprietà *LibraryName* e *VendorLib* ai nomi delle dlld associate. Una volta che Libraryname e *VendorLib* sono state impostate,l'applicazione non ha più bisogno di appoggiarsi a dbxdrivers.ini. (Ciò significa che non è necessario distribuire il file dbxdrivers.ini con l'applicazione, a meno che non si imposti la proprietà *DriverName* al momento dell'esecuzione.)

Specifica dei parametri di connessione

La proprietà *Params* è una lista di stringhe che elenca coppie di tipo nome/valore. Ogni coppia assume il formato *Name = Value*, dove *Name* è il nome del parametro e *Value* è il valore che gli si vuole assegnare.

I particolari parametri necessari dipendono dal server di database che si utilizza. Tuttavia, per tutti i server è richiesto un particolare parametro, *Database*. Il suo valore dipende dal server che si utilizza. Ad esempio, con InterBase, *Database* è il nome del file .gdb, con ORACLE è l'elemento in TNSNames.ora, mentre con DB2 è il nome del nodo sul lato client.

Altri parametri tipici includono lo *User_Name* (il nome da utilizzare per il login), *Password* (la password per *User_Name*), *HostName* (il nome della macchina o l'indirizzo IP dove è situato il server), e *TransIsolation* (il livello al cui raggiungimento le transazioni introdotte sono consapevoli delle modifiche apportate da altre transazioni). Quando si specifica un nome di driver, la proprietà *Params* viene precaricata con tutti i parametri necessari per quel tipo di driver, inizializzati ai valori predefiniti.

Poiché *Params* è una lista di stringhe, in progettazione è possibile fare doppio clic sulla proprietà *Params* nell'Object Inspector per modificare i parametri utilizzando lo String List editor. In esecuzione, utilizzare la proprietà *Params*::*Values* per assegnare i valori ai singoli parametri.

Assegnazione di un nome a una descrizione della connessione

Benché sia sempre possibile specificare una connessione utilizzando solo le proprietà *DatabaseName* e *Params*, può essere più conveniente assegnare un nome a una specifica combinazione e quindi identificare la connessione soltanto mediante il nome. È possibile assegnare un nome alle combinazioni di database di dbExpresse parametri, e di salvarle quindi in un file di nome dbxconnections.ini. Il nome di ogni combinazione viene detto nome della connessione.

Una volta che si è definito il nome della connessione, è possibile identificare una connessione di database semplicemente impostando la proprietà *ConnectionName* a un nome di connessione valido. L'impostazione di *ConnectionName* imposta automaticamente le proprietà *DriverName* e *Params*. Una volta impostato *ConnectionName*, è possibile modificare la proprietà *Params* per differenziarla temporaneamente rispetto al set salvato di valori di parametro; la modifica della proprietà *DriverName* azzera invece sia *Params* sia *ConnectionName*.

Uno dei vantaggi derivanti dall'utilizzo di nomi di connessione si presenta quando si sviluppa una applicazione utilizzando un certo database (ad esempio Local InterBase), ma la si distribuisce per utilizzarla con un altro (ad esempio ORACLE). In questo caso, *DriverName* e *Params* probabilmente saranno diversi sul sistema su cui si

distribuirà l'applicazione rispetto ai valori utilizzati durante lo sviluppo. È possibile scambiare le due descrizioni di connessione semplicemente utilizzando due versioni del file dbxconnections.ini. In progettazione, l'applicazione carica *DriverName* e *Params* dalla versione di progettazione di del file dbxconnections.ini. Quindi, quando si distribuirà l'applicazione, l'applicazione caricherà questi valori da una versione differente di dbxconnections.ini che utilizza il database "reale". Tuttavia, perché questo funzioni, è necessario fare in modo che il componente connection carichi di nuovo le proprietà *DriverName* e *Params*. È possibile farlo in due modi:

- Impostare la proprietà *LoadParamsOnConnect* a **true**. In questo modo si forza *TSQLConnection* a impostare automaticamente *DriverName* e *Params* ai valori associati a ConnectionNamenel file dbxconnections.ini non appena viene aperta la connessione.
- Chiamare il metodo *LoadParamsFromIniFile*. Questo metodo imposta *DriverName* e *Params* ai valori associati a *ConnectionName* nel file dbxconnections.ini(oppure in un altro file specificato). Si può scegliere di utilizzare questo metodo se si desiderano ridefinire alcuni valori di parametro prima di aprire la connessione.

Utilizzo di Connection Editor

Le relazioni tra i nomi di connessione e i rispettivi driver e parametri di connessione associati sono memorizzate nel file dbxconnections.ini. È possibile creare o modificare queste associazioni utilizzando Connection Editor.

Per visualizzare Connection Editor, fare doppio clic sul componente *TSQLConnection*. Verrà visualizzato Connection Editor che presenta una lista a discesa contenente tutti i driver disponibili, una lista di nomi di connessione per il driver attualmente selezionato e una tabella che elenca i parametri di connessione per il nome di connessione attualmente selezionato.

È possibile utilizzare questa finestra di dialogo per indicare la connessione da utilizzare selezionando un nome di driver e di connessione. Una volta scelta la configurazione desiderata, fare clic sul pulsante Test Connection per controllare che si sia scelta una configurazione valida.

Inoltre, è possibile utilizzare questa finestra di dialogo per modificare le connessioni con nome presenti in dbxconnections.ini:

- Modificare i valori dei parametri nella tabella dei parametri per modificare la connessione con nome attualmente selezionata. Appena si esce dalla finestra di dialogo facendo clic su OK, i nuovi valori dei parametri vengono salvati nel file dbxconnections.ini.
- Fare clic sul pulsante Add Connection per definire una connessione con nome. Verrà visualizzata una finestra di dialogo in cui si specificano il driver da utilizzare e il nome della nuova connessione. Una volta assegnato un nome alla connessione, modificare i parametri per specificare la connessione desiderata e fare clic sul pulsante OK per salvare la nuova connessione in dbxconnections.ini.
- Fare clic sul pulsante Delete Connection per cancellare da dbxconnections.ini la connessione con nome attualmente selezionata.

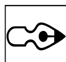
- Fare clic sul pulsante Rename Connection per modificare il nome della connessione con nome attualmente selezionata. Si noti che tutte le modifiche apportate ai parametri vengono salvate con il nuovo nome appena si fa clic sul pulsante OK.

Specifica dei dati da visualizzare

Esistono molti modi per specificare quali dati sono rappresentati da un dataset unidirezionale. Il metodo scelto dipende dal tipo di dataset unidirezionale utilizzato e dal fatto che le informazioni provengano da una singola tabella di database oppure siano il risultato di una query o di una procedura registrata.

Quando si opera con un componente *TSQLDataSet*, utilizzare la proprietà *CommandType* per indicare da dove il dataset ottiene i dati. *CommandType* può assumere uno dei seguenti valori:

- *ctQuery*: Se *CommandType* è *ctQuery*, *TSQLDataSet* esegue la query che è stata specifica. Se la query è un comando SELECT, il dataset contiene il set di record risultante.
- *ctTable*: Se *CommandType* è *ctTable*, *TSQLDataSet* ricerca tutti i record da una tabella specificata.
- *ctStoredProc*: Se *CommandType* è *ctStoredProc*, *TSQLDataSet* esegue una procedura registrata. Se la procedura registrata restituisce un cursor, il dataset contiene i record restituiti.



possibile anche riempire il dataset unidirezionale con i metadati relativi a tutto ciò che è disponibile sul server. Per informazioni sull'operazione, vedere "Prelievo di metadati in un dataset unidirezionale" a pagina 26-13.

Rappresentazione dei risultati di una query

L'utilizzo di una query è il metodo più generale per specificare un set di record. Le query sono semplicemente comandi scritti in SQL. Per rappresentare il risultato di una query è possibile utilizzare *TSQLDataSet* oppure *TSQLQuery*.

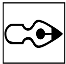
Quando si utilizza *TSQLDataSet*, impostare la proprietà *CommandType* a *ctQuery* e assegnare il testo dell'istruzione query alla proprietà *CommandText*. Quando si utilizza *TSQLQuery*, assegnare invece la query alla proprietà SQL. Queste proprietà funzionano allo stesso modo per tutti dataset di uso generale o di tipo query. Sono trattate in maggior dettaglio nel paragrafo. "Specifica della query" a pagina 22-44.

Quando si specifica la query, si possono includere parametri, o variabili, i cui valori possono essere modificati in progettazione o in esecuzione. I parametri possono sostituire i valori dei dati che appaiono nell'istruzione SQL. L'utilizzo di parametri nelle query e la fornitura di valori per quei parametri è trattato in "Utilizzo dei parametri nelle query" a pagina 22-46.

SQL definisce query che eseguono azioni sul server ma che non restituiscono record, come query di tipo UPDATE. Queste query sono trattate in "Esecuzione di comandi che non restituiscono record" a pagina 26-9.

Rappresentazione dei record in una tabella

Quando si devono rappresentare tutti i campi e tutti i record di una singola tabella di un database sottostante, invece di scrivere personalmente il codice SQL, per generare automaticamente la query è possibile utilizzare *TSQLDataSet* oppure *TSQLTable*.



Se le prestazioni del server possono essere un problema, si potrebbe comporre la query in modo esplicito invece di affidarsi a una query generata in automatico. Invece di elencare esplicitamente tutti i campi della tabella, le query generate in automatico utilizzano il carattere jolly. Questo comportamento causa sul server prestazioni leggermente più lente. L'utilizzo del carattere jolly (*) nelle query generate in automatico è più affidabile rispetto alla modifica dei campi sul server.

Rappresentazione di una tabella mediante TSQLDataSet

Per fare in modo che *TSQLDataSet* generi una query che recupera tutti i campi e tutti i record di una singola tabella di database, impostare la proprietà *CommandType* a *ctTable*.

Se si imposta *CommandType* a *ctTable*, *TSQLDataSet* genera una query in base ai valori di due proprietà:

- *CommandText* che specifica il nome della tabella di database che l'oggetto *TSQLDataSet* dovrebbe rappresentare.
- *SortFieldNames* che elenca i nomi di tutti i campi utilizzati per l'ordinamento dei dati, in ordine di importanza.

Ad esempio, se si specifica quanto segue:

```
SQLDataSet1->CommandType = ctTable;
SQLDataSet1->CommandText = "Employee";
SQLDataSet1->SortFieldNames = "HireDate,Salary"
```

TSQLDataSet genera la seguente query, che elenca tutti i record nella tabella Employee, ordinati per HireDate e, all'interno di HireDate, ordinati per Salary:

```
select * from Employee order by HireDate, Salary
```

Rappresentazione di una tabella mediante TSQLTable

Se si utilizza *TSQLTable*, specificare la tabella desiderata utilizzando la proprietà *TableName*.

Per specificare l'ordine dei campi nel dataset, è necessario specificare un indice. È possibile farlo in due modi:

- Impostando la proprietà *IndexName* al nome di un indice definito sul server che imponga l'ordine desiderato.

- Impostando la proprietà *IndexFieldNames* a una lista di nomi di campo, delimitata con caratteri punto e virgola, in base ai quali eseguire l'ordinamento. *IndexFieldNames* funziona come la proprietà SortFieldNamesdi *TSQLDataSet*, tranne per il fatto che utilizza come delimitatore un punto e virgola invece di una virgola.

Rappresentazione dei risultati di una procedura registrata

Le procedure registrate sono set di istruzioni SQL a cui è stato assegnato un nome e che è stato memorizzato su un server SQL. Il modo per indicare la procedura registrata che si desidera eseguire dipende dal tipo di dataset unidirezionale utilizzato.

Se si utilizza *TSQLDataSet*, per specificare una procedura registrata si deve:

- Impostare la proprietà *CommandType* a *ctStoredProc*.
- Specificare il nome della procedura registrata come valore della proprietà *CommandText*:

```
SQLDataSet1->CommandType = ctStoredProc;
SQLDataSet1->CommandText = "MyStoredProcName";
```

Se si utilizza *TSQLStoredProc* , è necessario specificare solo il nome della procedura registrata come valore della proprietà *StoredProcName*.

```
SQLStoredProc1->StoredProcName = "MyStoredProcName";
```

Dopo avere identificato una procedura registrata, l'applicazione potrebbe dover immettere dei valori per tutti i parametro di input della procedura registrata o recuperare i valori dei parametri di output dopo che si è eseguita la procedura registrata. Per informazioni sulle operazioni con i parametri delle procedure registrate, consultare "Operazioni con i parametri delle procedure registrate" a pagina 22-53.

Reperimento dei dati

Una volta specificata la sorgente dei dati, prima che l'applicazione possa accedere ai dati è necessario ottenerli dal server. Una volta che il dataset ha ottenuto i dati, i controlli associati ai dati collegati al dataset tramite una sorgente dati visualizzeranno automaticamente i valori dei dati, e sarà possibile riempire i dataset client, collegati al dataset tramite un provider, con i record.

Come con qualsiasi altro dataset, esistono due modi per ottenere i dati per un dataset unidirezionale:

- Impostare la proprietà *Active* a **true**, nell'Object Inspector in fase di progettazione oppure nel codice in fase di esecuzione:

  ```
  CustQuery->Active = true;
  ```

- Chiamare il metodo *Open* in fase di esecuzione:

  ```
  CustQuery->Open();
  ```

Utilizzare la proprietà *Active* o il metodo *Open* con qualsiasi dataset unidirezionale che ottenga i record dal server. Non ha importanza se questi record derivano da una query SELECT (incluse le query generate in automatico nel caso in cui *CommandType* sia impostato *ctTable*) o da una procedura registrata.

Preparazione del dataset

Affinché una query o una procedura registrata possa essere eseguita sul server, è necessario prima "prepararla". Preparare il dataset significa che *dbExpress* e il server devono allocare risorse per l'istruzione e per i relativi parametri. Se *CommandType* è impostata a *ctTable*, questo è il momento in cui il dataset genera la query SELECT. A questo punto, tutti i parametri che sono scollegati dal server vengono inglobati in una query.

I dataset unidirezionali vengono preparati automaticamente quando si imposta *Active* a **true** o si chiama il metodo *Open*. Quando si chiude il dataset, le risorse allocate per l'esecuzione dell'istruzione vengono liberate. Se si pensa di eseguire la query o la procedura registrata più di una volta, è possibile migliorare le prestazioni preparando esplicitamente il dataset prima di aprirlo per la prima volta. Per preparare esplicitamente un dataset, impostare la proprietà *Prepared* del dataset a **true**.

```
CustQuery->Prepared = true;
```

Quando si prepara esplicitamente il dataset, le risorse allocate per l'esecuzione dell'istruzione non vengono liberate finché non si imposta *Prepared* a **false**.

Impostare la proprietà *Prepared* a false se si desidera essere certi che il dataset venga preparato di nuovo prima dell'esecuzione (ad esempio, se si modifica un valore di parametro o la proprietà *SortFieldNames*).

Reperimento di più dataset

Alcune procedure registrate restituiscono più set di record.Appena lo si apre, il dataset recupera solo il primo set. Per accedere agli altri set di record, chiamare il metodo *NextRecordSet*:

```
TCustomSQLDataSet *DataSet2 = SQLStoredProc1->NextRecordSet(nRows);
```

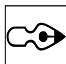
NextRecordSet restituisce un componente *TCustomSQLDataSet* creato ex novo che consente l'accesso al successivo set di record. Vale a dire, la prima volta che si chiama *NextRecordSet*, il metodo restituisce un dataset per il secondo set di record.La successiva chiamata a *NextRecordSet* restituisce un terzo dataset e così via, finché non vi sono più altri set di record. Quando non vi sono più altri dataset, *NextRecordSet* restituisce NULL.

Esecuzione di comandi che non restituiscono record

È possibile utilizzare un dataset unidirezionale anche se la query o la procedura registrata che esso rappresenta non restituisce alcun record. Tali comandi includono

istruzioni di tipo DDL (Data Definition Language) o DML (Data Manipulation Language) diverse da istruzioni SELECT (ad esempio, i comandi INSERT, DELETE, UPDATE, CREATE INDEX e ALTER TABLE non restituiscono alcun record). Il linguaggio utilizzato nei comandi è specifico del server, ma di solito conforme allo standard SQL-92 per il linguaggio SQL.

Il comando SQL che si esegue deve essere conforme alle specifiche del server utilizzato. I dataset unidirezionali non sono in grado né di valutare né di eseguire l'istruzione SQL. Essi si limitano solo a passare il comando al server affinché venga eseguito.



Se il comando non restituisce alcun record, non è affatto necessario utilizzare un dataset unidirezionale, in quanto non c'è alcuna necessità di utilizzare i metodi del dataset per accedere a un set di record. Il componente SQL connection che si collega al server di database può essere utilizzato per eseguire direttamente un comando sul server. Per maggiori informazioni, consultare "Invio di comandi al server" a pagina 21-11.

Specifica del comando da eseguire

Con i dataset unidirezionali esiste un solo modo per specificare il comando da eseguire, indipendentemente dal fatto che il comando restituisca o meno un dataset. Vale a dire:

Se si utilizza *TSQLDataSet*, utilizzare le proprietà *CommandType* e *CommandText* per specificare il comando:

- Se *CommandType* è *ctQuery*, *CommandText* è l'istruzione SQL da passare al server.
- Se *CommandType* è *ctStoredProc*, *CommandText* è il nome della procedura registrata da eseguire.

Se si utilizza *TSQLQuery*, utilizzare la proprietà *SQL* per specificare l'istruzione SQL da passare al server.

Se si utilizza *TSQLStoredProc*, utilizzare la proprietà *StoredProcName* per specificare il nome della procedura registrata da eseguire.

Dato che si specifica il comando in modo analogo a quanto si fa per reperire i record, si opererà con i parametri della query o della procedura registrata in un modo analogo a quello usato con le query e con le procedure registrate che restituiscono record. Per maggiori informazioni, consultare "Utilizzo dei parametri nelle query" a pagina 22-46 e "Operazioni con i parametri delle procedure registrate" a pagina 22-53.

Esecuzione del comando

Per eseguire una query o una procedura registrata che non restituisce alcun record, non si utilizza la proprietà *Active* o il metodo *Open*. Si deve invece utilizzare

- Il metodo *ExecSQL* se il dataset è un'istanza di *TSQLDataSet* o di *TSQLQuery*.

```
FixTicket->CommandText = "DELETE FROM TrafficViolations WHERE (TicketID = 1099)";
```

```
FixTicket->ExecSQL();
```

- Il metodo *ExecProc* se il dataset è un'istanza di *TSQLStoredProc*.

```
SQLStoredProc1->StoredProcName = "MyCommandWithNoResults";
SQLStoredProc1->ExecProc();
```



Se si esegue più volte la query o la procedura registrata, è conveniente impostare la proprietà *Prepared* a **true**.

Creazione e modifica dei metadati del server

Quasi tutti i comandi che non restituiscono dati rientrano in due categorie: comandi che di solito si utilizzano per modificare i dati (come i comandi INSERT, DELETE e UPDATE) e comandi che di solito vengono utilizzati per creare o modificare le entità sul server, come tabelle, indici e procedure registrate.

Se non si vogliono utilizzare comandi SQL espliciti per l'editing, è possibile collegare il dataset unidirezionale a un dataset client e lasciare che sia il dataset a gestire la generazione di tutti i comandi SQL relativi all'editing (vedere "Connessione di un dataset client a un altro dataset nella stessa applicazione" a pagina 18-12). Infatti, questo è l'approccio consigliato dal momento che i controlli associati ai dati sono progettati per eseguire l'editing tramite un dataset come *TClientDataSet*.

L'unico modo che ha l'applicazione per creare o modificare i metadati sul server, tuttavia, è quello di inviare un comando. Non tutti i driver di database supportano la stessa sintassi SQL. La descrizione della sintassi SQL supportata da ogni tipo di database e delle differenze fra i tipi di database travalica lo scopo di questo manuale. Per una discussione completa e aggiornata dell'implementazione SQL per un determinato sistema di database, consultare la documentazione fornita con quel sistema.

Di solito, si utilizza l'istruzione CREATE TABLE per creare le tabelle in un database e l'istruzione CREATE INDEX per creare nuovi indici per quelle tabelle. Se supportate, si possono utilizzare altre istruzioni CREATE per aggiungere diversi oggetti metadati, come CREATE DOMAIN, CREATE VIEW, CREATE SCHEMA e CREATE PROCEDURE.

Per ognuna delle istruzioni CREATE, esiste un'istruzione DROP corrispondente per cancellare l'oggetto metadati. Queste istruzioni includono DROP TABLE, DROP VIEW, DROP DOMAIN, DROP SCHEMA e DROP PROCEDURE.

Per modificare la struttura di una tabella, utilizzare l'istruzione ALTER TABLE. ALTER TABLE ha clausole ADD e DROP per creare nuovi elementi in una tabella e per cancellarli. Ad esempio, utilizzare la clausola ADD COLUMN per aggiungere alla tabella una nuova colonna e DROP CONSTRAINT per cancellare dalla tabella un vincolo esistente.

Ad esempio, la seguente istruzione crea una procedura registrata di nome GET_EMP_PROJ in un database InterBase:

```
CREATE PROCEDURE GET_EMP_PROJ (EMP_NO SMALLINT)
RETURNS (PROJ_ID CHAR(5))
AS
```

```
BEGIN
  FOR SELECT PROJ_ID
  FROM EMPLOYEE_PROJECT
  WHERE EMP_NO = :EMP_NO
  INTO :PROJ_ID
  DO
    SUSPEND;
END
```

Il codice seguente utilizza un *TSQLDataSet* per creare questa procedura registrata. Si noti dell'utilizzo della proprietà *ParamCheck* per impedire al dataset di confondere i parametri nella definizione della procedura registrata (:EMP_NO e :PROJ_ID) con un parametro della query che crea la procedura registrata.

```
SQLDataSet1->ParamCheck = false;
SQLDataSet1->CommandType = ctQuery;
SQLDataSet1->CommandText = "CREATE PROCEDURE GET_EMP_PROJ (EMP_NO SMALLINT) RETURNS (PROJ_ID
CHAR(5)) AS BEGIN FOR SELECT PROJ_ID FROM EMPLOYEE_PROJECT WHERE EMP_NO = :EMP_NO INTO
:PROJ_ID DO SUSPEND; END";
SQLDataSet1->ExecSQL();
```

Configurazione di cursor master/detail collegati

Esistono due modi per utilizzare cursor collegati al fine di impostare una relazione master/detail che utilizza un dataset unidirezionale per il set di dettaglio. Il metodo utilizzato dipende dal tipo di dataset unidirezionale che si utilizza. Una volta impostata tale relazione, il dataset unidirezionale (la parte "molti" in una relazione uno-a-molti) fornisce l'accesso solo a quei record che corrispondono al record corrente del set master (la parte "uno" in una relazione uno-a-molti).

TSQLDataSet e *TSQLQuery* richiedono che si utilizzi una query con parametri per stabilire una relazione master/detail. Questa è la tecnica per la creazione di tali relazioni su tutti i dataset di tipo query. Per i dettagli sulla creazione delle relazioni master/detail con dataset di tipo query, vedere "Impostazione di relazioni master/ detail mediante parametri" a pagina 22-49.

Per impostare una relazione master/detail dove il set di dettaglio è un'istanza di *TSQLTable*, utilizzare le proprietà *MasterSource* e *MasterFields* esattamente come si farebbe con qualsiasi altro dataset di tipo tabella. Per i dettagli sulla creazione delle relazioni master/detail con dataset di tipo tabella, vedere "Impostazione di relazioni master/detail mediante parametri" a pagina 22-49.

Accesso a informazioni di schema

Esistono due modi per ottenere informazioni su ciò che è disponibile sul server. Queste informazioni, dette informazioni di schema o metadati, includono informazioni su quali tabelle e procedure registrate sono disponibili sul server e informazioni su quelle tabelle e su quelle procedure registrate (come i campi contenuti in una tabella, gli indici definiti e i parametri utilizzati da una procedura registrata).

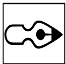
Il modo più semplice per ottenere questi metadati è quello di utilizzare i metodi di *TSQLConnection*. Questi metodi riempiono una lista di stringhe esistente oppure un oggetto list con i nomi di tabelle, di procedure registrate, di campi o di indici o con i descrittori di parametro. Questa tecnica è analoga a quella utilizzata con tutti gli altri componenti di connessione database per riempire le liste con metadati. Questi metodi sono descritti in "Ottenimento di metadati" a pagina 21-14.

Se si richiedono informazioni di schema più dettagliate, è possibile riempire un dataset unidirezionale con metadati. Invece di riempire una semplice lista, il dataset unidirezionale vine riempito con le informazioni di schema, in cui ogni record rappresenta una singola tabella, una procedura registrata, un indice, un campo o un parametro.

Prelievo di metadati in un dataset unidirezionale

Per riempire i dataset unidirezionali con i metadati ottenuti dal server di database, è necessario prima indicare quali dati si desidera vedere utilizzando il metodo *SetSchemaInfo*. *SetSchemaInfo* accetta tre parametri:

- Il tipo di informazioni di schema (metadati) che si desidera ottenere. Questo può essere una lista di tabelle (*stTables*), una lista di tabelle di sistema (*stSysTables*), una lista di procedure registrate (*stProcedures*), una lista di campi in una tabella (*stColumns*), una lista di indici (*stIndexes*) o una lista dei parametri utilizzati da una procedura registrata (*stProcedureParams*). Ogni tipo di informazioni utilizza un set diverso di campi per descrivere gli elementi nella lista. Per i dettagli sulle strutture di questi dataset, vedere "La struttura dei dataset di metadati" a pagina 26-14.
- Se si vogliono ottenere informazioni su campi, indici o parametri di una procedura registrata, il nome della tabella o della procedura registrata a cui tali informazioni si applicano. Se si vogliono ottenere altri tipi di informazioni di schema, questo parametro è NULL.
- Un modello che deve trovare una corrispondenza per ogni nome restituito. Questo modello è un modello SQL come 'Cust%', che utilizza i caratteri jolly '%' (che corrisponde a una stringa di caratteri arbitrari di qualsiasi lunghezza) e '_' (che corrisponde a un singolo carattere arbitrario). Per utilizzare nel modello un simbolo percentuale o un carattere di sottolineatura, si deve raddoppiare il carattere (%% o __). Se non si desidera utilizzare un modello, questo parametro può essere NULL.



Se si vogliono ottenere informazioni di schema sulle tabelle (*stTables*), le informazioni di schema risultanti possono descrivere tabelle comuni, tabelle di sistema, viste e/o sinonimi, a seconda del valore della proprietà *TableScope* della connessione SQL.

La seguente chiamata richiede una tabella che elenca tutte le tabelle di sistema (tabelle del server che contengono metadati):

```
SQLDataSet1->SetSchemaInfo(stSysTable, "", "");
```

Se si apre il dataset dopo questa chiamata a *SetSchemaInfo*, il dataset risultante contiene un record per ogni tabella, con colonne che riportano il nome di tabella, il

tipo, il nome dello schema, e così via. Se il server non utilizza tabelle di sistema per memorizzare i metadati (ad esempio MySQL), il dataset non conterrà alcun record.

Nell'esempio precedente si è utilizzato solo il primo parametro. Si supponga, invece, di voler ottenere una lista di parametri di input per una procedura registrata di nome 'MyProc'. Si supponga, inoltre, che chi ha scritto quella procedura registrata abbia chiamato tutti i parametri utilizzando un prefisso per indicare se erano parametri di input o di output ('inName', 'outValue' e così via). In questo caso, si potrebbe chiamare *SetSchemaInfo* in questo modo:

```
SQLDataSet1->SetSchemaInfo(stProcedureParams, "MyProc", "in%");
```

Il dataset risultante è una tabella di parametri di input con colonne che descrivono le proprietà di ogni parametro.

Ottenimento dei dati dopo aver utilizzato il dataset per i metadati

Dopo aver effettuato una chiamata a *SetSchemaInfo*, esistono due modi per ritornare con il dataset alle query o alle procedure registrate in esecuzione:

- Modificare la proprietà *CommandText*, specificando la query, la tabella o la procedura registrata da cui si desidera ottenere i dati.
- Chiamare *SetSchemaInfo*, impostando il primo parametro a *stNoSchema*. In questo caso, il dataset ritorna a prendere i dati specificati dal valore corrente di *CommandText*.

La struttura dei dataset di metadati

Per ogni tipo di metadati a cui è possibile accedere utilizzando *TSQLDataSet*, esiste un insieme predefinito di colonne (campi) che viene compilato con le informazioni sugli elementi del tipo richiesto.

Informazioni sulle tabelle

Quando si richiedono informazioni sulle tabelle (*stTables* o *stSysTables*), il dataset risultante include un record per ciascuna tabella. Ha le seguenti colonne:

Tabella 26.1 Colonne nelle tabelle di metadati che elencano tabelle

Nome della colonna	Tipo di campo	Contenuto
RECNO	ftInteger	Un numero di record che identifica in modo univocamente ciascun record.
CATALOG_NAME	ftString	Il nome del catalogo (database) che contiene la tabella. È analogo al parametro *Database* di un componente connection SQL.
SCHEMA_NAME	ftString	Il nome dello schema che identifica il proprietario della tabella.

Tabella 26.1 Colonne nelle tabelle di metadati che elencano tabelle

Nome della colonna	Tipo di campo	Contenuto
TABLE_NAME	ftString	Il nome della tabella. Questo campo determina la sequenza di ordinamento del dataset.
TABLE_TYPE	ftInteger	Identifica il tipo di tabella. È la somma di uno o più dei seguenti valori: 1: Tabella 2: Vista 4: Tabella di sistema 8: Sinonimo 16: Tabella temporanea 32: Tabella locale.

Informazioni sulle procedure registrate

Quando si richiedono informazioni sulle procedure registrate (*stProcedures*), il dataset risultante include un record per ogni procedura registrata. Ha le seguenti colonne:

Tabella 26.2 Colonne nelle tabelle di metadati che elencano procedure registrate

Nome della colonna	Tipo di campo	Contenuto
RECNO	ftInteger	Un numero di record che identifica in modo univocamente ciascun record.
CATALOG_NAME	ftString	Il nome del catalogo (database) che contiene la procedura registrata. È analogo al parametro *Database* di un componente connection SQL.
SCHEMA_NAME	ftString	Il nome dello schema che identifica il proprietario della procedura registrata.
PROC_NAME	ftString	Il nome della procedura registrata. Questo campo determina la sequenza di ordinamento del dataset.
PROC_TYPE	ftInteger	Identifica il tipo di procedura registrata. È la somma di uno o più dei seguenti valori: 1: Procedura (nessun valore restituito) 2: Funzione (restituisce un valore) 4: Package 8: Procedura di sistema
IN_PARAMS	ftSmallint	Il numero di parametri di input
OUT_PARAMS	ftSmallint	Il numero di parametri di output.

Informazioni sui campi

Quando si richiedono informazioni sui campi in una tabella specificata (*stColumns*), il dataset risultante include un record per ciascun campo. Include le seguenti colonne:

Tabella 26.3 Colonne nelle tabelle di metadati che elencano campi

Nome della colonna	Tipo di campo	Contenuto
RECNO	ftInteger	Un numero di record che identifica in modo univocamente ciascun record.
CATALOG_NAME	ftString	Il nome del catalogo (database) che contiene la tabella di cui si stanno elencando i campi. È analogo al parametro *Database* di un componente connection SQL.
SCHEMA_NAME	ftString	Il nome dello schema che identifica il proprietario del campo.
TABLE_NAME	ftString	Il nome della tabella che contiene i campi.
COLUMN_NAME	ftString	Il nome del campo. Questo valore determina la sequenza di ordinamento del dataset.
COLUMN_POSITION	ftSmallint	La posizione della colonna nella rispettiva tabella.
COLUMN_TYPE	ftInteger	Identifica il tipo di valore nel campo. È la somma di uno o più dei seguenti valori: 1: Identificativo della riga 2: Versione della riga 4: Campo ad incremento automatico 8: Campo con valore predefinito
COLUMN_DATATYPE	ftSmallint	Il tipo di dati della colonna. Questa è una delle costanti logiche di tipo field definite in sqllinks.h.
COLUMN_TYPENAME	ftString	Una stringa che descrive il tipo di dati. Queste sono le stesse informazioni contenute in COLUMN_DATATYPE e in COLUMN_SUBTYPE, ma nel formato utilizzato in alcune istruzioni DDL.
COLUMN_SUBTYPE	ftSmallint	Un sottotipo per il tipo di dati della colonna. Questa è una delle costanti logiche di tipo subtype definite in sqllinks.h.
COLUMN_PRECISION	ftInteger	La dimensione del tipo di campo (numero di caratteri in una stringa, byte in uno campo Byte, cifre significative in un valore BCD, membri di un campo ADT, e così via).
COLUMN_SCALE	ftSmallint	Il numero di cifre alla destra della virgola per valori BCD, o discendenti su ADT e campi array.
COLUMN_LENGTH	ftInteger	Il numero di byte richiesti per memorizzare i valori del campo.
COLUMN_NULLABLE	ftSmallint	Un Boolean che indica se il campo può essere lasciato in bianco (0 significa che il campo richiede un valore).

Informazioni sugli indici

Quando si richiedono informazioni sugli indici su una tabella (*stIndexes*), il dataset risultante include un record per ogni campo in ciascun record. (Gli indici multi-

record vengono descritti utilizzando più record):Tabella 26.4 Colonne nelle tabelle di metadati che elencano indici:

Tabella 26.4 Colonne nelle tabelle di metadati che elencano indici

Nome della colonna	Tipo di campo	Contenuto
RECNO	ftInteger	Un numero di record che identifica in modo univocamente ciascun record.
CATALOG_NAME	ftString	Il nome del catalogo (database) che contiene l'indice. È analogo al parametro *Database* di un componente connection SQL.
SCHEMA_NAME	ftString	Il nome dello schema che identifica il proprietario dell'indice.
TABLE_NAME	ftString	Il nome della tabella per cui l'indice è definito.
INDEX_NAME	ftString	Il nome dell'indice. Questo campo determina la sequenza di ordinamento del dataset.
PKEY_NAME	ftString	Indica il nome della chiave primaria.
COLUMN_NAME	ftString	Il nome del campo (colonna) nell'indice.
COLUMN_POSITION	ftSmallint	La posizione di questo campo nell'indice.
INDEX_TYPE	ftSmallint	Identifica il tipo di indice. È la somma di uno o più dei seguenti valori: 1: Non-univoco 2: Univoco 4: Chiave primaria
SORT_ORDER	ftString	Indica che l'indice è ascendente (a) o discendente (d).
FILTER	ftString	Descrive una condizione di filtro che limita i record indicizzati.

Informazioni sui parametri delle procedure registrate

Quando si richiedono informazioni sui parametri di una procedura registrata (*stProcedureParams*), il dataset risultante include un record per ogni parametro. Ha le seguenti colonne:

Tabella 26.5 Colonne nelle tabelle di metadati che elencano parametri

Nome della colonna	Tipo di campo	Contenuto
RECNO	ftInteger	Un numero di record che identifica in modo univocamente ciascun record.
CATALOG_NAME	ftString	Il nome del catalogo (database) che contiene la procedura registrata. È analogo al parametro *Database* di un componente connection SQL.
SCHEMA_NAME	ftString	Il nome dello schema che identifica il proprietario della procedura registrata.
PROC_NAME	ftString	Il nome della procedura registrata che contiene il parametro.
PARAM_NAME	ftString	Il nome del parametro. Questo campo determina la sequenza di ordinamento del dataset.

Tabella 26.5 Colonne nelle tabelle di metadati che elencano parametri

Nome della colonna	Tipo di campo	Contenuto
PARAM_TYPE	ftSmallint	Identifica il tipo di parametro. È analogo alla proprietà *ParamType* di un oggetto *TParam*.
PARAM_DATATYPE	ftSmallint	Il tipo di dati del parametro. Questa è una delle costanti logiche di tipo field definite in sqllinks.h.
PARAM_SUBTYPE	ftSmallint	Un sottotipo per il tipo di dati del parametro. Questa è una delle costanti logiche di tipo subtype definite in sqllinks.h.
PARAM_TYPENAME	ftString	Una stringa che descrive il tipo di dati. Queste sono le stesse informazioni contenute in PARAM_DATATYPE e in PARAM_SUBTYPE, ma in un formato utilizzato in alcune istruzioni DDL.
PARAM_PRECISION	ftInteger	Il numero massimo di cifre nei valori a virgola mobile o di byte (per stringhe e campi Bytes).
PARAM_SCALE	ftSmallint	Il numero di cifre alla destra della virgola in valori a virgola mobile.
PARAM_LENGTH	ftInteger	Il numero di byte richiesti per memorizzare i valori del parametro.
PARAM_NULLABLE	ftSmallint	Un Boolean che indica se il parametro può essere lasciato in bianco (0 significa che il parametro richiede un valore).

Debug di applicazioni dbExpress

Mentre si esegue il debug dell'applicazione database, potrebbe rivelarsi utile controllare i messaggi SQL che vengono inviati e ricevuti dal server di database attraverso il componente connection, inclusi quelli generati automaticamente (ad esempio da un componente provider o dal driver *dbExpress*).

Utilizzo di TSQLMonitor per controllare i comandi SQL

TSQLConnection utilizza un componente di appoggio, *TSQLMonitor*, per intercettare questi messaggi e salvarli in una lista di stringhe. *TSQLMonitor* opera in modo simile al programma di utilità SQL monitor che è possibile utilizzare con BDE, tranne per il fatto che controlla solo quei comandi che implicano un singolo componente *TSQLConnection* invece di tutti i comandi gestiti da *dbExpress*.

Per utilizzare *TSQLMonitor*:

1 Aggiungere un componente *TSQLMonitor* alla scheda o al modulo dati che contiene il componente *TSQLConnection* di cui si desidera controllare i comandi SQL.

2 Impostare la proprietà *SQLConnection* al componente *TSQLConnection*.

3 Impostare la proprietà *Active* di SQL monitor a **true**.

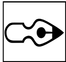
Man mano che i comandi SQL vengono inviati al server, la proprietà *TraceList* di SQL monitor viene aggiornata automaticamente elencando tutti i comandi SQL intercettati.

È possibile salvare questa lista in un file specificando un valore per la proprietà *FileName* e quindi impostando la proprietà *AutoSave* a **true**. *AutoSave* fa in modo che SQL monitor salvi in un file il contenuto della proprietà *TraceList* ogni volta che intercetta un nuovo messaggio.

Se non si desidera l'appesantimento dovuto al salvataggio del file ogni volta che viene intercettato un messaggio, è possibile utilizzare il gestore di evento *OnLogTrace* per salvare il file solo dopo l'intercettazione di vari messaggi. Ad esempio, il seguente gestore di evento salva il contenuto di *TraceList* ogni 10 messaggi, cancellando il file di registrazione dopo averlo salvato in modo che la lista non divenga mai troppo lunga:

```
void __fastcall TForm1::SQLMonitor1LogTrace(TObject *Sender, void *CBInfo)
{
  TSQLMonitor *pMonitor = dynamic_cast<TSQLMonitor *>(Sender);
  if (pMonitor->TraceCount == 10)
  {
    // build unique file name
    AnsiString LogFileName = "c:\\log";
    LogFileName = LogFileName + IntToStr(pMonitor->Tag);
    LogFileName = LogFileName + ".txt"
    pMonitor->Tag = pMonitor->Tag + 1;
    // Save contents of log and clear the list
    pMonitor->SaveToFile(LogFileName);
    pMonitor->TraceList->Clear();

}
```



Se si decidesse di utilizzare il gestore di evento precedente, alla chiusura dell'applicazione sarebbe necessario salvare anche una eventuale lista parziale (contenente meno di 10 voci.

Utilizzo di una callback per controllare i comandi SQL

Invece di utilizzare *TSQLMonitor*, è possibile personalizzare il modo in cui l'applicazione tiene traccia dei comandi SQL utilizzando il metodo *SetTraceCallbackEvent* del componente SQL connection. *SetTraceCallbackEvent* accetta due parametri: una callback di tipo *TSQLCallbackEvent* e un valore definito dall'utente che viene passato alla funzione callback.

La funzione callback accetta due parametri: *CallType* e *CBInfo*:

- *CallType* è un parametro riservato per un successivo utilizzo.
- *CBInfo* è un puntatore a un structure che include la categoria (la stessa di *CallType*), il testo del comando SQL e il valore definito dall'utente che viene passato al metodo *SetTraceCallbackEvent*.

La callback restituisce un valore di tipo *CBRType*, di solito *cbrUSEDEF*.Il driver *dbExpress* chiama la callback ogni volta che il componente SQL connection passa un comando al server o il server restituisce un messaggio di errore.



Non chiamare *SetTraceCallbackEvent* se all'oggetto *TSQLConnection* è associato un componente *TSQLMonitor*. *TSQLMonitor* utilizza il meccanismo di callback per operare e *TSQLConnection* può supportare una sola callback per volta.

Capitolo

27

Utilizzo dei dataset client

I dataset client sono dataset specializzati che conservano tutti i propri dati in memoria. Il supporto per il trattamento dei dati che essi memorizzano in memoria è fornito da midas.dll. Il formato utilizzato dai dataset client per la memorizzazione di dati è indipendente ed è facilmente trasportabile, il che consente ai dataset client di

- Leggere e scrivere in file dedicati su disco, agendo come un dataset basato su file. Le proprietà e i metodi che supportano questo meccanismo sono descritti in "Uso di un dataset client con dati basati su file" a pagina 27-35.
- Memorizzare in cache gli aggiornamenti per i dati provenienti da un server di database. Le funzioni dei dataset client che supportano gli aggiornamenti in cache sono descritte in "Utilizzo di un dataset client per registrare in cache gli aggiornamenti" a pagina 27-17.
- Rappresentare i dati nella parte client di un'applicazione multi-tier. Per funzionare in questo modo, il dataset client deve operare con un provider esterno, come descritto in "Uso di un dataset client con un provider di dati" a pagina 27-26. Per informazioni sulle applicazioni database multi-tier, consultare il Capitolo 29, "Creazione di applicazioni multi-tier".
- Rappresentare i dati da una sorgente diversa da un dataset. Poiché un dataset client può utilizzare i dati provenienti da un provider esterno, i provider specializzati possono adattare una varietà di fonti di informazioni per operare con i dataset client. Ad esempio, è possibile utilizzare un provider XML per consentire a un dataset client di rappresentare le informazioni contenute in un documento XML.

Sia che si utilizzino i dataset client per dati basati su file, per memorizzare in cache gli aggiornamenti, per dati provenienti da un provider esterno (come nel caso di operazioni con un documento XML o in un'applicazione multi-tier) o per una combinazione di questi approcci come nel caso di un'applicazione "modello briefcase", è possibile sfruttare i vantaggi offerti dalla vasta gamma di funzioni supportate dai dataset client per operare con i dati.

Utilizzo dei dati tramite un dataset client

Come si fa con qualunque dataset, è possibile utilizzare i dataset client per fornire i dati per i controlli associati ai dati ricorrendo a un componente sorgente dati. Per informazioni su come visualizzare i dati del database nei controlli data-aware, consultare il Capitolo 19, "Uso dei controlli dati".

I dataset client implementano tutte le proprietà e i metodi ereditati da *TDataSet*. Per una presentazione completa di questo comportamento generico dei dataset, consultare il Capitolo 22, "I dataset".

Inoltre, i dataset client implementano molte delle funzioni comuni ai dataset di tipo tabella come:

- Ordinamento dei record con indici.
- Utilizzo di indici per la ricerca di record.
- Limitazione dei record con gli intervalli.
- Creazione di relazioni master/detail.
- Controllo dell'accesso in lettura/scrittura
- Creazione del dataset sottostante
- Svuotamento del dataset
- Sincronizzazione dei dataset client

Per i dettagli relativi a queste funzioni, consultare "Utilizzo dei dataset di tipo tabella" a pagina 22-26.

I dataset client differiscono dagli altri dataset in quanto mantengono tutti i loro dati in memoria. Pertanto, il loro supporto per alcune funzioni di database possono comportare ulteriori capacità o considerazioni. Questo capitolo descrive alcune di queste funzioni comuni e le differenze introdotte dai dataset client.

Come spostarsi tra i dati nei dataset client

Se un'applicazione utilizza controlli associati ai dati standard, un utente può spostarsi tra i record di un dataset client usando il comportamento nativo di questi controlli. È possibile spostarsi tra i record da programma utilizzando i metodi standard dei dataset, quali *First*, *Last*, *Next* e *Prior*. Per ulteriori informazioni su questi metodi, consultare "Spostamenti nei dataset" a pagina 22-5.

Diversamente dalla maggior parte dei dataset, i dataset client possono anche posizionare il cursore su un record specifico del dataset utilizzando la proprietà *RecNo*. Generalmente, un'applicazione usa *RecNo* per determinare il numero di record del record attivo. I dataset client possono, tuttavia, impostare *RecNo* a un particolare numero di record per far sì che quel record diventi attivo.

Come limitare i record da visualizzare

To Per limitare temporaneamente l'accesso da parte degli utenti a un sottogruppo di dati disponibili, le applicazioni possono utilizzare gli intervalli e i filtri. Quando si applica un intervallo o un filtro, il dataset client non visualizza tutti i dati presenti

nella sua memoria cache. Mostra, invece, solo i dati che soddisfano le condizioni dell'intervallo o del filtro. Per ulteriori informazioni sull'utilizzo di filtri, consultare "Visualizzazione e modifica di un sottoinsieme di dati usando i filtri" a pagina 22-13. Per ulteriori informazioni sugli intervalli, consultare "Limitazione dei record con gli intervalli" a pagina 22-32.

Con la maggior parte dei dataset le stringhe del filtro vengono suddivise in comandi SQL che vengono poi implementati nel database server. Ne deriva che il dialetto SQL del server limita le operazioni che possono essere utilizzate nelle stringhe del filtro. I dataset client implementano un proprio supporto per i filtri, che include un maggior numero di operazioni rispetto agli altri dataset. Per esempio, quando si utilizza un dataset client, le espressioni filtro possono includere operatori stringa che restituiscono sottostringhe, operatori che suddividono i valori data/ora, ed altro ancora. I dataset client consentono, inoltre, l'uso di filtri sui campi BLOB o su tipi di campi complessi come i campi ADT e i campi array.

Nella tabella seguente sono elencati gli operatori e le funzioni che i dataset client possono utilizzare nei filtri, oltre a un confronto con altri dataset che supportano filtri:

Tabella 27.1 Supporto per i filtri nei dataset client

Operatore o funzione	Esempio	Supportato dagli altri dataset	Commento
Confronto			
=	State = 'CA'	Sì	
<>	State <> 'CA'	Sì	
>=	DateEntered >= '1/1/1998'	Sì	
<=	Total <= 100,000	Sì	
>	Percentile > 50	Sì	
<	Field1 < Field2	Sì	
BLANK	State <> 'CA' or State = BLANK	Sì	I record vuoti non vengono visualizzati a meno che non siano esplicitamente inclusi nel filtro.
IS NULL	Field1 IS NULL	No	
IS NOT NULL	Field1 IS NOT NULL	No	
Operatori logici			
and	State = 'CA' and Country = 'US'	Sì	
or	State = 'CA' or State = 'MA'	Sì	
not	not (State = 'CA')	Sì	
Operatori aritmetici			
+	Total + 5 > 100	Dipende dal driver	Vale per numeri, stringhe o date (ore) + numero.

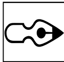
Tabella 27.1 Supporto per i filtri nei dataset client (continua)

Operatore o funzione	Esempio	Supportato dagli altri dataset	Commento
-	Field1 - 7 <> 10	Dipende dal driver	Vale per numeri, stringhe o date (ore) - numero.
*	Discount * 100 > 20	Dipende dal driver	Vale solo per numeri.
/	Discount > Total / 5	Dipende dal driver	Vale solo per numeri.
Funzioni di stringa			
Upper	Upper(Field1) = 'ALWAYS'	No	
Lower	Lower(Field1 + Field2) = 'josp'	No	
Substring	Substring(DateFld,8) = '1998' Substring(DateFld,1,3) = 'JAN'	No	Il valore va dalla posizione del secondo argomento fino alla fine o fino al numero di caratteri specificato nel terzo argomento. Il primo carattere ha la posizione 1.
Trim	Trim(Field1 + Field2) Trim(Field1, '-')	No	Rimuove il terzo argomento dalla parte iniziale e finale. Se non c'è un terzo argomento, elimina gli spazi.
TrimLeft	TrimLeft(StringField) TrimLeft(Field1, '$') <> ''	No	Vedi Trim.
TrimRight	TrimRight(StringField) TrimRight(Field1, '.') <> ''	No	Vedi Trim.
Funzioni DateTime			
Year	Year(DateField) = 2000	No	
Month	Month(DateField) <> 12	No	
Day	Day(DateField) = 1	No	
Hour	Hour(DateField) < 16	No	
Minute	Minute(DateField) = 0	No	
Second	Second(DateField) = 30	No	
GetDate	GetDate - DateField > 7	No	Rappresenta la data e l'ora correnti.
Date	DateField = Date(GetDate)	No	Restituisce la parte data di un valore datetime.
Time	TimeField > Time(GetDate)	No	Restituisce la parte ora di un valore datetime.

Tabella 27.1 Supporto per i filtri nei dataset client (continua)

Operatore o funzione	Esempio	Supportato dagli altri dataset	Commento
Varie			
Like	Memo LIKE '%filters%'	No	Funziona come in SQL -92 senza la clausola ESC. Se applicato a campi BLOB, FilterOptions determina se viene tenuto conto di maiuscole e minuscole.
In	Day(DateField) in (1,7)	No	Funziona come in SQL-92. Il secondo argomento è una lista di valori tutto dello stesso tipo.
*	State = 'M*'	Sì	Carattere jolly per confronti parziali.

Quando si applicano intervalli o filtri, il dataset client memorizza anche tutti i suoi record in memoria. L'intervallo o il filtro determina semplicemente quali record sono disponibili per i controlli che servono per visualizzare o per spostarsi tra i dati del dataset client.



Quando si recuperano dati da un provider è possibile anche limitare i dati memorizzati nel dataset client fornendo dei parametri al provider. Per ulteriori informazioni, consultare "Limitazione dei record mediante parametri" a pagina 27-31.

Editing di dati

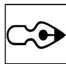
I dataset client rappresentano i propri dati come un pacchetto dati in memoria. Questo pacchetto è il valore della proprietà *Data* del dataset client. Per impostazione predefinita, tuttavia, le modifiche non vengono memorizzate nella proprietà *Data*. Invece, gli inserimenti, le cancellazioni e le modifiche (apportate dagli utenti o da programma) vengono memorizzati in un registro interno delle modifiche, rappresentato dalla proprietà *Delta*. L'uso di un registro delle modifiche serve a due scopi:

- Il registro delle modifiche è necessario per applicare gli aggiornamenti a un server di database o a un componente provider esterno.
- Il registro delle modifiche fornisce un supporto sofisticato per l'annullamento delle modifiche.

La proprietà *LogChanges* consente di disattivare temporaneamente la registrazione. Quando *LogChanges* è **true**, le modifiche vengono memorizzate nel registro. Quando è **false**, le modifiche vengono apportate direttamente alla proprietà *Data*. È possibile disattivare il registro delle modifiche nelle applicazioni basate su file se il supporto per l'annullamento non è necessario.

Le modifiche rimangono nel registro finché non vengono rimosse dall'applicazione. Le applicazioni rimuovono le modifiche quando

- Vengono annullate
- Vengono salvate



Il salvataggio del dataset client in un file non rimuove le modiche dal registro delle modifiche. Quando si ricarica il dataset, le proprietà *Data* e *Delta* rimangono identiche a com'erano quando i dati sono stati salvati.

Annullamento delle modifiche

Anche se la versione originale di un record rimane invariata in *Data*, ogni volta che un utente modifica un record, lo lascia e vi ritorna, l'utente vede la versione modificata per ultima di un record. Se un utente o un'applicazione modifica un record un certo numero di volte, ogni versione modificata del record viene memorizzata nel registro delle modifiche come una voce separata.

La memorizzazione di ciascuna modifica apportata a un record rende possibile il supporto di più livelli di operazioni di annullamento di quante ne occorrerebbero per ripristinare lo stato precedente di un record:

- Per rimuovere l'ultima modifica apportata a un record, occorre chiamare *UndoLastChange*. *UndoLastChange* accetta un parametro booleano, *FollowChange*, che indica se riposizionare o no il cursore sul record ripristinato (**true**), o se lasciare il cursore sul record corrente (**false**). Se a un record sono state apportate diverse modifiche, ogni chiamata a *UndoLastChange* rimuove un livello di modifiche. *UndoLastChange* restituisce un valore booleano che indica la riuscita o meno dell'operazione. Se la rimozione ha successo, *UndoLastChange* restituisce **true**. Per determinare se ci sono ulteriori modifiche da annullare, si deve ricorrere alla proprietà *ChangeCount*. *ChangeCount* indica il numero di modifiche memorizzate nel registro delle modifiche.
- Anziché rimuovere ogni livello di modifica apportata a un singolo record, è possibile rimuoverle tutti contemporaneamente. Per rimuovere tutte le modifiche di un record, occorre selezionarlo e chiamare *RevertRecord*. *RevertRecord* rimuove dal registro delle modifiche tutti i cambiamenti apportati al record attivo.
- Per ripristinare un record cancellato, per prima cosa impostare la proprietà *StatusFilter* a [*usDeleted*], il che rende " visibili" i record cancellati. Quindi, spostarsi sul record che si desidera ripristinare e chiamare *RevertRecord*. Infine, ripristinare la proprietà *StatusFilter* a [*usModified*, *usInserted*, *usUnmodified*] in modo che la versione modificata del dataset (ora contenente il record ripristinato) sia nuovamente visibile.
- In qualunque punto durante le modifiche è possibile salvare lo stato attuale del registro delle modifiche utilizzando la proprietà *SavePoint*. La lettura di *SavePoint* restituisce un marcatore nella posizione corrente del registro delle modifiche. In seguito, se si vogliono annullare tutte modifiche apportate dopo la lettura del punto in cui è stato effettuato il salvataggio, occorre impostare *SavePoint* al valore letto in precedenza. L'applicazione può ottenere valori per diversi punti di salvataggio. Tuttavia, una volta effettuata la copia di riserva del registro delle

modifiche in un punto di salvataggio, i valori di tutti i punti di salvataggio letti successivamente dall'applicazione non sono validi.

- Chiamando *CancelUpdates*, si possono abbandonare tutte le modifiche memorizzate nel registro. *CancelUpdates* azzera il registro delle modifiche, rimuovendo di fatto tutte le modifiche apportate ai record. *CancelUpdates* va, pertanto, utilizzato con molta attenzione.Dopo aver chiamato *CancelUpdates*, non è possibile recuperare le modifiche che si trovavano nel registro.

Salvataggio delle modifiche

I dataset client utilizzano meccanismi diversi per incorporare le modifiche dal registro delle modifiche, a seconda del fatto che i dataset client registrino i propri dati in un file rappresentino dei dati ottenuti mediante un provider. Indipendentemente dal meccanismo usato, il registro delle modifiche viene svuotato automaticamente quando tutti gli aggiornamenti sono stati incorporati.

Le applicazioni autonome possono semplicemente fondere le modifiche nella cache locale rappresentata dalla proprietà *Data*. Esse non devono preoccuparsi di come risolvere i problemi derivanti dalle modifiche locali con quelle apportate da altri utenti. Per fondere il registro delle modifiche nella proprietà *Data*, occorre chiamare il metodo *MergeChangeLog*. Il paragrafo "Fusione delle modifiche nei dati" a pagina 27-36 descrive questa procedura.

È impossibile utilizzare *MergeChangeLog* se si utilizza il dataset client per memorizzare in cache gli aggiornamenti o per rappresentare i dati da un componente provider esterno. Le informazioni nel registro delle modifiche sono necessarie per la risoluzione dei record aggiornati con i dati memorizzati nel database (o nel dataset sorgente). In alternativa, si può chiamare *ApplyUpdates*, che tenta di scrivere le modifiche nel database server (o nel dataset sorgente) e aggiorna la proprietà *Data* solo quando le modifiche sono state inoltrate con successo. Per ulteriori informazioni su questa procedura, consultare "Applicazione degli aggiornamenti" a pagina 27-22.

Definizione dei vincoli dei valori dati

I dataset client possono imporre dei vincoli alle modifiche apportabili da un utente ai dati. Questi vincoli vengono applicati non appena l'utente tenta di registrare le modifiche nel registro delle modifiche. È sempre possibile fornire vincoli personalizzati. In questo modo è possibile preparare dei propri vincoli, definiti dall'applicazione, sui valori che gli utenti possono inviare a un dataset client.

Inoltre, quando i dataset client rappresentano dati di un server a cui si accede mediante BDE, essi impongono anche i vincoli sui dati importati dal server di database. Se il dataset client opera con un componente provider esterno, il provider può controllare se quei vincoli vengono inviati al dataset client e il dataset client può controllare se li utilizza. Per ulteriori informazioni su come il provider controlla se i vincoli sono inclusi nei pacchetti dati, consultare "Gestione dei vincoli del server" a pagina 28-13. Per i dettagli su come e perché il dataset client può disattivare l'imposizione dei vincoli del server, consultare "Gestione dei vincoli dal server" a pagina 27-32.

Specifica di vincoli custom

È possibile utilizzare le proprietà dei componenti campo del dataset client per imporre dei vincoli sui dati che l'utente può immettere. Ogni componente campo ha due proprietà che è possibile utilizzare per specificare i vincoli:

- La proprietà *DefaultExpression* definisce un valore predefinito che viene assegnato al campo nel caso in cui l'utente non immetta un valore. Si noti che se anche il database server, o il dataset sorgente, assegna al campo un'espressione predefinita, la versione del dataset client ha la precedenza perché viene assegnata prima che l'aggiornamento sia applicato al database server o al dataset sorgente.
- La proprietà *CustomConstraint* consente di assegnare una condizione restrittiva che deve essere soddisfatta prima di poter confermare il valore del campo. I vincoli custom definiti in questo modo vengono imposti in aggiunta a tutte le restrizioni importate dal server. Per ulteriori informazioni sulle operazioni con i vincoli custom sui componenti campo, consultare "Creazione di un vincolo custom" a pagina 23-22.

In aggiunta, tramite la proprietà *Constraints* del dataset client, è possibile creare anche vincoli a livello di record. *Constraints* è una raccolta di oggetti *TCheckConstraint*, in cui ciascun oggetto rappresenta una condizione differente. Utilizzare la proprietà *CustomConstraint* di un oggetto *TCheckConstraint* per aggiungere i propri vincoli che saranno controllati al momento della registrazione dei record.

Ordinamento e indicizzazione

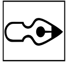
L'uso degli indici fornisce diversi vantaggi alle applicazioni:

- Consente ai dataset client di individuare rapidamente i dati.
- Permettono di applicare gli intervalli per limitare i record disponibili.
- Permettono alle applicazioni di impostare relazioni con altri dataset come tabelle di consultazione o schede master/detail.
- Consente di specificare l'ordine in cui devono apparire i record.

Se un dataset client rappresenta i dati di un server o utilizza un provider esterno, eredita un indice e un criterio di ordinamento predefinito in base ai dati che riceve. L'indice predefinito si chiama DEFAULT_ORDER. Si può utilizzare questo ordinamento, ma non è possibile cambiare o cancellare l'indice.

Oltre all'indice predefinito, il dataset client mantiene un secondo indice, denominato CHANGEINDEX, sui record modificati nel registro delle modifiche (proprietà *Delta*). CHANGEINDEX ordina tutti i record del dataset client come apparirebbero se venissero applicate le modifiche specificate in *Delta*. CHANGEINDEX si basa sull'ordinamento ereditato da DEFAULT_ORDER. Come avviene con DEFAULT_ORDER, non è possibile cambiare o cancellare l'indice CHANGEINDEX.

È possibile utilizzare altri indici esistenti o crearne di propri. Nelle sezioni seguenti viene descritta la procedura per la creazione e l'uso degli indici con i dataset client.



Potrebbe essere opportuno riesaminare gli argomenti relativi agli indici nei dataset di tipo tabella, validi anche nel caso dei dataset client. Gli argomenti sono trattati nelle sezioni "Ordinamento dei record con indici" a pagina 22-27 e "Limitazione dei record con gli intervalli" a pagina 22-32.

Aggiunta di un nuovo indice

Esistono due modi per aggiungere indici ad un dataset client:

- Per creare in fase di esecuzione un indice temporaneo che ordini i record del dataset client è possibile utilizzare la proprietà *IndexFieldNames*. Indicare i nomi dei campi, separandoli con il carattere punto e virgola. La sequenza con cui sono indicati i campi nell'elenco determina il loro ordine nell'indice.

  Questo è il metodo meno efficiente per aggiungere indici. Non è possibile specificare indici discendenti, o che tengano conto di maiuscole e minuscole, e l'indice risultante non supporta il raggruppamento. Una volta chiuso il dataset, questi indici non sono permanenti e non vengono salvati se si salva il dataset client in un file.

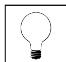
- Per creare in fase di esecuzione un indice utilizzabile per il raggruppamento, chiamare il metodo *AddIndex*. *AddIndex* consente di specificare le proprietà dell'indice, compresi:

  - Il nome dell'indice. Questo può essere utilizzato per cambiare gli indici in fase di esecuzione.
  - I campi che costituiscono l'indice. L'indice utilizza tali campi per ordinare i record e per individuare i record che hanno valori specifici in quei campi.
  - La modalità secondo la quale l'indice ordina i record. Per impostazione definita, gli indici impongono un ordinamento crescente (basato sulla macchina locale). Questo ordinamento predefinito tiene conto della differenza tra lettere maiuscole e minuscole. È possibile specificare delle opzioni che facciano sì che l'intero indice ignori la differenza tra maiuscole e minuscole o che ordinino in modo discendente. In alternativa, è possibile fornire una lista di campi da ordinare senza tener conto delle maiuscole\minuscole e una lista di campi da ordinare in ordine discendendente.
  - Il livello predefinito di raggruppamento supportato per l'indice.

  Gli indici creati con *AddIndex* non permangono quando si chiude il dataset client. (Ovvero, vanno persi quando si riapre il dataset client). Non è possibile chiamare *AddIndex* quando il dataset è chiuso. Gli indici che vengono aggiunti tramite *AddIndex* non vengono salvati quando si salva il dataset client in un file.

- Il terzo modo per creare un indice consiste nel crearlo al momento della creazione del dataset client. Prima di creare il dataset client, specificare gli indici desiderati utilizzando la proprietà *IndexDefs*. Gli indici vengono creati in un secondo momento insieme al dataset associato quando si chiamerà *CreateDataSet*. Per ulteriori informazioni sulla creazione di dataset client, consultare "Creazione e cancellazione di tabelle" a pagina 22-40.

Come con *AddIndex*, gli indici creati con il raggruppamento del supporto del dataset possono eseguire l'ordinamento ascendente su alcuni campi e discendente su altri, e possono tenere conto delle maiuscole/minuscole su alcuni campi e non su altri. Gli indici così creati permangono sempre e vengono salvati quando si salva il dataset client in un file.



L'indicizzazione e l'ordinamento possono essere effettuati sono sui campi calcolati internamente con i dataset client.

Cancellazione e cambio degli indici

Per rimuovere un indice creato per un dataset client, occorre chiamare *DeleteIndex* e specificare il nome dell'indice da rimuovere. Non è possibile cancellare gli indici DEFAULT_ORDER e CHANGEINDEX.

Per usare un indice diverso quando sono disponibili più indici, si deve ricorrere alla proprietà *IndexName* per selezionare l'indice da utilizzare. In progettazione, si può operare la scelta dagli indici disponibili nella casella a discesa della proprietà *IndexName* dell'Object Inspector.

Uso degli indici per il raggruppamento dei dati

Quando si usa un indice nel dataset client, questo impone automaticamente un tipo di ordinamento sui record. Dato questo ordinamento, i record adiacenti generalmente contengono valori duplicati nei campi che costituiscono l'indice. Per esempio, si consideri il seguente frammento di una tabella ordini indicizzata sui campi SalesRep e Customer:

SalesRep	Customer	OrderNo	Amount
1	1	5	100
1	1	2	50
1	2	3	200
1	2	6	75
2	1	1	10
2	3	4	200

Dato questo ordinamento, i valori adiacenti nella colonna SalesRep sono duplicati. Lo stesso avviene all'interno dei record di SalesRep 1, i cui valori adiacenti della colonna Customer sono duplicati.Cioè, i dati sono raggruppati per SalesRep e all'interno del gruppo SalesRep sono raggruppati per Customer. Ogni raggruppamento ha un livello associato. In questo caso il gruppo SalesRep ha il livello 1 (in quanto non è annidato in nessun altro gruppo), mentre il gruppo Customer ha come livello 2 (in quanto è annidato nel gruppo con il livello 1). Il livello di raggruppamento corrisponde all'ordine dei campi nell'indice.

I dataset client consentono di determinare dove sta il record attivo all'interno di un determinato livello di raggruppamento. Ciò consente all'applicazione di visualizzare i record in modo differente, a seconda che si tratti del primo record del gruppo, della parte centrale di un gruppo o dell'ultimo record di un gruppo. Per esempio, si potrebbe voler visualizzare il valore di un campo solo se è il primo record del

gruppo, eliminando i valori duplicati. Questa operazione con la tabella precedente produce il seguente risultato:

SalesRep	Customer	OrderNo	Amount
1	1	5	100
		2	50
	2	3	200
		6	75
2	1	1	10
	3	4	200

Per determinare dove cade il record attivo all'interno di un gruppo, si deve ricorrere al metodo *GetGroupState*. *GetGroupState* accetta un integer corrispondente al livello del gruppo e restituisce un valore che indica dove cade il valore corrente nel gruppo (primo record, secondo record o nessuno dei due).

Quando si crea un indice, si può specificare il livello di raggruppamento supportato (fino al numero dei campi dell'indice). *GetGroupState* non è in grado di fornire informazioni sui gruppi oltre quel livello, anche se l'indice ordina i record utilizzando campi aggiuntivi.

Rappresentazione dei valori calcolati

Come con qualunque dataset, è possibile aggiungere campi calcolati al dataset client. Questi sono campi i cui valori vengono calcolati in modo dinamico, generalmente sulla base dei valori di altri campi dello stesso record. Per ulteriori informazioni sull'uso dei campi calcolati, consultare "Definizione di un campo calcolato" a pagina 23-7.

I dataset client, tuttavia, consentono l'ottimizzazione quando i campi vengono calcolati utilizzando i campi calcolati internamente. Per maggiori informazioni sui campi calcolati internamente, consultare il paragrafo successivo "Uso di campi calcolati internamente nei dataset client".

È anche possibile comunicare ai dataset client di creare campi calcolati che sommino i dati di diversi record utilizzando le aggregazioni di manutenzione. Per ulteriori informazioni sulla aggregazioni di manutenzione, consultare "Uso delle aggregazioni di manutenzione" a pagina 27-12.

Uso di campi calcolati internamente nei dataset client

Negli altri dataset l'applicazione deve calcolare il valore dei campi calcolati ogni volta che cambia il record o l'utente modifica dei campi del record attivo. Ciò avviene in un gestore di evento *OnCalcFields*.

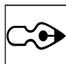
Anche se è ancora possibile procedere in questo modo, i dataset client consentono di ridurre al minimo il numero di volte in cui i campi calcolati devono essere ricalcolati, salvandoli nei dati del dataset client. Quando i valori calcolati vengono salvati con il dataset client, essi devono ancora essere ricalcolati quando l'utente modifica il record

attivo, ma l'applicazione non richiede il ricalcolo dei valori ogni volta che cambia il record attivo. Per salvare i valori calcolati nei dati del dataset client, si devono utilizzare i campi calcolati ulteriormente al posto dei campi calcolati.

I campi calcolati internamente, esattamente come i campi calcolati, vengono elaborati in un gestore di evento *OnCalcFields*. Tuttavia, è possibile ottimizzare il gestore di evento contrassegnando la proprietà *State* del dataset client. Quando *State* è *dsInternalCalc*, occorre ricalcolare i campi calcolati internamente. Quando *State* è *dsCalcFields*, occorre ricalcolare i campi calcolati normali.

Per usare i campi calcolati internamente, occorre definire i campi come tali prima di creare il dataset client. A seconda che si utilizzino i campi persistenti o le definizioni di campo, l'operazione viene svolta in uno dei seguenti modi:

- Se si utilizzano campi persistenti, si devono definire i campi come calcolati internamente selezionando InternalCalc nel Fields editor.
- Se si utilizzano le definizioni dei campi, impostare a **true** la proprietà *InternalCalcField* della definizione rilevante del campo.



Altri tipi di dataset utilizzano campi calcolati internamente. Tuttavia, con gli altri dataset questi valori non vengono calcolati in un gestore di evento *OnCalcFields*. Invece, vengono calcolati automaticamente tramite da BDE o da un server di database remoto.

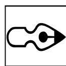
Uso delle aggregazioni di manutenzione

I dataset client forniscono il supporto per il riepilogo dei dati provenienti da gruppi di record. Dal momento che questi riepiloghi vengono aggiornati automaticamente quando si modificano i dati del dataset, questi dati riepilogati vengono definiti una "aggregazione di manutenzione."

Nella loro forma più semplice le aggregazioni di manutenzione consentono di ottenere informazioni come la somma di tutti i valori di una colonna del dataset client. Esse sono, comunque, sufficientemente flessibili per supportare un grande numero di calcoli di somma e di fornire totali parziali di gruppi di record definiti in un indice che supporta il raggruppamento.

Specifica delle aggregazioni

Per specificare che si vogliono calcolare riepiloghi di record di un dataset client, si deve ricorrere alla proprietà *Aggregates*. *Aggregates* è una raccolta di specifiche di aggregazione (*TAggregate*). È possibile aggiungere tali specifiche al dataset client utilizzando il Collection Editor in progettazione o il metodo *Add* di *Aggregates* in esecuzione. Se si vogliono creare componenti campo per le aggregazioni, occorre creare campi persistenti per i valori di aggregazione nel Fields Editor.



Quando si creano campi di aggregazione, alla proprietà *Aggregates* del dataset client vengono aggiunti automaticamente gli oggetti appropriati di aggregazione. Non occorre aggiungerli in modo esplicito durante la creazione dei campi persistenti di aggregazione. Per ulteriori dettagli sulla creazione dei campi persistenti di aggregazione, consultare "Definizione di un campo di aggregazione" a pagina 23-10.

Per ciascuna aggregazione la proprietà *Expression* indica il calcolo di riepilogo che rappresenta. *Expression* può contenere una semplice espressione di riepilogo come

```
Sum(Field1)
```

oppure un'espressione complessa che abbina informazioni di campi diversi, come

```
Sum(Qty * Price) - Sum(AmountPaid)
```

Le espressioni di aggregazione includono uno o più degli operatori di riepilogo riportati nella Tabella 27.2

Tabella 27.2 Operatori di riepilogo per le aggregazioni di manutenzione

Operatore	Uso
Sum	Calcola il totale dei valori di un campo numerico o di un'espressione
Avg	Calcola il valore mezzo di un campo numerico o data-ora oppure di un'espressione
Count	Specifica il numero di valori non vuoti per un campo o un'espressione
Min	Indica il valore minimo per un campo stringa, numerico o data-ora oppure per un'espressione
Max	Indica il valore massimo per un campo stringa, numerico o data-ora oppure per un'espressione

Gli operatori di riepilogo agiscono sui valori dei campi o sulle espressioni costruite dai valori dei campi utilizzando gli stessi operatori usati per creare i filtri.(In ogni caso non si possono annidare operatori di riepilogo). È possibile creare espressioni utilizzando operatori su valori di riepilogo con altri valori di riepilogo, o su valori di riepilogo e costanti. Tuttavia, non è consentito combinare valori di riepilogo e valori di campo, in quanto tali espressioni sono ambigue (non ci sono indicazioni su quale record deve fornire il valore del campo). Queste regole vengono illustrate nelle espressioni seguenti:

```
Sum(Qty * Price)               {consentito - riepilogo di un'espressione su campi }
Max(Field1) - Max(Field2)      {consentito - espressione su riepiloghi }
Avg(DiscountRate) * 100        {consentito - espressione di riepilogo e costante }
Min(Sum(Field1))               {non consentito - riepiloghi annidati }
Count(Field1) - Field2         {non consentito -- espressione di riepilogo e campo }
```

Aggregazione di gruppi di record

Per impostazione predefinita, le aggregazioni di manutenzione vengono calcolate in modo che effettuino il riepilogo di tutti i record nel dataset client. Tuttavia, si può anche specificare che si vuole effettuare il riepilogo sui record di un gruppo. Ciò consente di fornire riepiloghi intermedi, come dei totali parziali per gruppi di record che condividono un valore di campo comune.

Prima che sia possibile specificare un'aggregazione di manutenzione su un gruppo di record, bisogna ricorrere a un indice che supporti il raggruppamento appropriato. Per informazioni sul supporto per il raggruppamento, consultare "Uso degli indici per il raggruppamento dei dati" a pagina 27-10.

Una volta che si è in possesso di un indice che raggruppa in dati nel modo che si vuole che vengano riepilogati, occorre specificare le proprietà *IndexName* e *GroupingLevel* dell'aggregazione per indicare quale indice utilizza e quale gruppo o sottogruppo di quel indice definisce i record da riepilogare.

Per esempio, si consideri il frammento seguente di una tabella ordini che è raggruppare per SalesRep e, all'interno di SalesRep, per Customer:

SalesRep	Customer	OrderNo	Amount
1	1	5	100
1	1	2	50
1	2	3	200
1	2	6	75
2	1	1	10
2	3	4	200

Il codice seguente imposta un'aggregazione di manutenzione che indica la quantità complessiva per ogni rivenditore:

```
Agg->Expression = "Sum(Amount)";
Agg->IndexName = "SalesCust";
Agg->GroupingLevel = 1;
Agg->AggregateName = "Total for Rep";
```

Per aggiungere un'aggregazione che effettui il riepilogo per ogni cliente all'interno di un determinato rivenditore, occorre creare un'aggregazione di manutenzione con livello 2.

Le aggregazioni di manutenzione che effettuano il riepilogo su un gruppo di record vengono associate con un indice specifico. La proprietà *Aggregates* può includere aggregazioni che utilizzano indici diversi. Tuttavia, solo le aggregazioni che effettuano il riepilogo sull'intero dataset e quelle che utilizzano l'indice corrente sono valide. Il cambiamento dell'indice corrente cambia le aggregazioni valide. Per determinare quali aggregazioni sono valide in qualunque momento, bisogna ricorrere alla proprietà *ActiveAggs*.

Ottenimento dei valori di aggregazione

Per ottenere il valore di un'aggregazione di manutenzione, occorre chiamare il metodo *Value* dell'oggetto *TAggregate* che rappresenta l'aggregazione. *Value* restituisce l'aggregazione di manutenzione del gruppo che contiene il record corrente del dataset client.

Quando si effettua il riepilogo dell'intero dataset client, si può chiamare in qualunque momento *Value* per ottenere l'aggregazione di manutenzione. Tuttavia, quando si effettua il riepilogo su informazioni raggruppate, occorre verificare che il record attivo si trovi nel gruppo che si intende riepilogare. Pertanto, è consigliabile ottenere valori di aggregazione in tempi chiaramente specificati, come quando si passa sul primo record di un gruppo o ci si sposta sull'ultimo. Per determinare dove cade il record attivo all'interno di un gruppo, si deve utilizzare il metodo *GetGroupState*.

Per visualizzare le aggregazioni di manutenzione nei controlli associati ai dati, si deve utilizzare il Fields editor per creare un componente campo persistente di aggregazione. Quando si specifica un campo di aggregazione nel Fields editor, *Aggregates* del dataset client viene aggiornato automaticamente, in modo da includere la specifica di aggregazione appropriata. La proprietà *AggFields* contiene il nuovo componente campo di aggregazione, che viene restituito dal metodo *FindField*.

Copia di dati da un altro dataset

Per copiare in progettazione i dati da un altro dataset, fare clic con il pulsante destro sul dataset e scegliere Assign Local *Data*. Apparirà una finestra di dialogo con l'elenco di tutti i dataset disponibili nel progetto. Selezionare quello di cui si vogliono copiare dati e struttura e scegliere OK. Quando si copia il dataset sorgente, viene attivato automaticamente il dataset client.

Per copiare dati da un altro dataset in fase di esecuzione, è possibile assegnare direttamente i dati, oppure, se il sorgente è un altro dataset client, clonare il cursor.

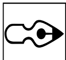
Assegnazione diretta dei dati

La proprietà *Data* del dataset client può essere utilizzato per assegnare a un dataset client i dati di un altro dataset. *Data* è un pacchetto dati sotto forma di OleVariant. Un pacchetto dati può derivare da un altro dataset client, o da qualunque altro dataset utilizzando un provider. Una volta che un pacchetto dati è stato assegnato a *Data*, il suo contenuto viene visualizzato automaticamente nei controlli associati ai dati connessi al dataset client tramite un componente datasource.

Quando si apre un dataset client che rappresenta i dati di un server o che utilizza un componente provider esterno, i pacchetti dati vengono assegnati automaticamente a *Data*.

Quando il dataset client non usa un provider, è possibile copiare i dati da un altro dataset nel modo seguente:

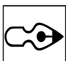
```
ClientDataSet1->Data = ClientDataSet2->Data;
```



Quando si copia la proprietà *Data* di un altro dataset client, si copia anche il registro delle modifiche, ma la copia non riflette i filtri o gli intervalli che sono stati applicati. Per includere i filtri o gli intervalli, occorre clonare il cursor del dataset sorgente.

Se si effettua la copia da un dataset alternativo rispetto ad un dataset client, si può creare un componente provider del dataset, collegarlo al dataset sorgente e poi copiarne i dati:

```
TempProvider = new TDataSetProvider(Form1);
TempProvider->DataSet = SourceDataSet;
ClientDataSet1->Data = TempProvider->Data;
delete TempProvider;
```



Quando si assegnano direttamente i dati alla proprietà *Data*, il nuovo pacchetto dati viene fuso nei dati esistenti. Tutti i dati precedenti vengono, invece, sostituiti.

Se si intendono fondere le modifiche ricavandole da un altro dataset, anziché copiarne i dati, si deve utilizzare un componente provider. Creare un provider di

dataset come nell'esempio precedente, ma associarlo al dataset di destinazione e, anziché copiare la proprietà *Data*, utilizzare il metodo *ApplyUpdates*:

```
TempProvider = new TDataSetProvider(Form1);
TempProvider->DataSet = ClientDataSet1;
TempProvider->ApplyUpdates(SourceDataSet->Delta, -1, ErrCount);
delete TempProvider;
```

Clonazione di un cursor di un dataset client

I dataset client utilizzano il metodo *CloneCursor* per permettere di gestire in fase di esecuzione una seconda vista dei dati. *CloneCursor* consente a un secondo dataset client di condividere i dati del dataset client originale. Ciò è meno oneroso della copia di tutti i dati originali, ma, dal momento che i dati sono condivisi, il secondo dataset client non può modificare i dati senza influenzare il dataset client originale.

CloneCursor richiede tre parametri: *Source* specifica il dataset client da clonare. Gli ultimi due parametri (*Reset* e *KeepSettings*) indicano se copiare informazioni diverse dai dati. Queste informazioni comprendono tutti i filtri, l'indice attivo, i collegamenti a una tabella master (quando il dataset sorgente è un set di dettaglio), la proprietà *ReadOnly* e tutti i collegamenti a un componente connessione o a un interfaccia provider.

Quando *Reset* e *KeepSettings* sono **false**, viene aperto un dataset client clonato e vengono utilizzate le impostazioni del dataset sorgente per definire le proprietà della destinazione. Quando *Reset* è **true**, alle proprietà del dataset di destinazione vengono attribuiti i valori predefiniti (non viene specificato nessun indice o filtro, nessuna tabella master, nessun componente connessione o provider, e *ReadOnly* è **false**). Quando *KeepSettings* è **true**, le proprietà del dataset di destinazione non vengono modificate.

Aggiunta di informazioni specifiche per l'applicazione ai dati

Gli sviluppatori di applicazioni possono aggiungere informazioni personalizzate alla proprietà *Data* del dataset client. Dal momento che queste informazioni vengono fornite insieme al pacchetto dati, esse vengono incluse quando si salvano i dati in un file o in un flusso. Esse vengono copiate quando si copia i dati in un altro dataset. Opzionalmente, possono essere incluse con la proprietà *Delta* in modo che un provider possa leggerle quando riceve gli aggiornamenti dai dataset client.

Per salvare con la proprietà *Data* informazioni specifiche per l'applicazione, si deve usare il metodo *SetOptionalParam*. Questo consente di memorizzare un OleVariant che contiene i dati con un nome specifico.

Per recuperare queste informazioni specifiche dell'applicazione, si deve ricorrere al metodo *GetOptionalParam*, passando il nome utilizzato quando sono state memorizzate le informazioni.

Utilizzo di un dataset client per registrare in cache gli aggiornamenti

Per impostazione predefinita, quando si modificano i dati nella maggior parte dei dataset, ogni volta che si cancella o si registra un record, il dataset genera una transazione, cancella o scrive quel record nel server di database e convalida la transazione. Se nello scrivere le modifiche nel database si verifica un problema, l'applicazione viene immediatamente informata: il dataset solleva un'eccezione quando si registra il record.

Se il dataset utilizza un server di database remoto, questo approccio può degradare le prestazioni a causa del traffico di rete tra l'applicazione e il server ogni volta che ci si sposta a un nuovo record dopo avere modificato il record corrente. Per minimizzare il traffico di rete, potrebbe essere opportuno memorizzare gli aggiornamenti nella cache locale.Quando si memorizzano gli aggiornamenti nella cache, l'applicazione preleva i dati dal database, li memorizza nella cache e lo modifica in locale e quindi applica gli aggiornamenti in cache al database con una singola transazione. Se si utilizzano gli aggiornamenti in cache, le modifiche a un dataset (come la registrazione delle modifiche o la cancellazione di record) vengono memorizzate in locale invece di essere scritte direttamente nella tabella associata al dataset. Una volta completate le modifiche, l'applicazione chiama un metodo che scrive nel database le modifiche memorizzate nella cache e azzera la cache.

Gli aggiornamenti in cache possono ridurre al minimo la durata della transazione e limitare il traffico nella rete.Tuttavia, i dati nella cache sono locali rispetto all'applicazione e non sono sotto il controllo della transazione. Tuttavia, i dati nella cache sono locali all'applicazione e non sono sotto il controllo della transazione. Ciò significa che mentre si lavora sulla propria copia locale in memoria dei dati, altre applicazioni potrebbero modificare i dati nella tabella di database sottostante. Inoltre, non sono in grado di vedere alcuna modifica finché gli aggiornamenti memorizzati in cache non saranno applicati. Per questo motivo, l'uso degli aggiornamenti in cache potrebbe non essere appropriato per applicazioni che operano con dati volatili, dal momento che si potrebbero causare o incontrare troppi conflitti non appena si cercherà di fondere le modifiche nel database.

Benché BDE e ADO forniscano meccanismi alternativi per la memorizzazione in cache degli aggiornamenti, l'utilizzo di un dataset client a tale scopo presenta diversi vantaggi:

- L'applicazione degli aggiornamenti quando i dataset sono collegati mediante relazioni master/detail è gestita automaticamente. Questo fatto garantisce che gli aggiornamenti a più dataset collegati vengano applicati nell'ordine corretto.
- I dataset client offrono il massimo controllo sul processo di aggiornamento. È possibile impostare delle proprietà per influire sulle istruzioni SQL generate per l'aggiornamento di record, per specificare la tabella da utilizzare durante l'aggiornameto di record provenienti da una join su più tabelle, o addirittura applicare gli aggiornamenti manualmente mediante da un gestore di evento *BeforeUpdateRecord*.

- Se durante l'applicazione al server di database degli aggiornamenti in cache si verificano degli errori, solo i dataset client (e i provider di dataset) sono in grado di fornire informazioni sul valore attuale del record nel server di database, oltre al valore originale (non modificato) presente nel dataset e al nuovo valore (modificato) dell'aggiornamento che non è andato a buon fine.
- I dataset client permettono di specificare il numero di errori di aggiornamento che si desidera tollerare prima che l'intero aggiornamento venga annullato.

Panoramica sull'utilizzo degli aggiornamenti in cache

Per utilizzare gli aggiornamenti in cache, le varie fasi elaborative devono avvenire in un'applicazione secondo l'ordine seguente:

1 **Indicare i dati che si vogliono modificare**. La modalità operativa dipende dal tipo di dataset client utilizzato:

- Se si utilizza *TClientDataSet*, specificare il componente provider che rappresenta i dati che si vogliono modificare. L'operazione è descritta in "Specifica di un provider" a pagina 27-27.
- Se si utilizza un dataset client associato a un particolare meccanismo di accesso di dati, è necessario
  - Identificare il server di database impostando la proprietà *DBConnection* a un componente connection appropriato.
  - Indicare quali dati si vogliono vedere specificando le proprietà CommandTexte *CommandType*. La proprietà *CommandType* indica se *CommandText* rappresenta un'istruzione SQL da eseguire, il nome di una procedura registrata oppure il nome di una tabella. Se *CommandText* è una query o una procedura registrata, utilizzare la proprietà *Params* per fornire tutti i parametri di input.
  - Facoltativamente, utilizzare la proprietà *Options* per indicare se i set di dettaglio annidati e i dati BLOB devono essere inclusi nei pacchetti dati o recuperati separatamente, se determinati tipi di modifiche (inserimenti, modifiche o cancellazioni) devono essere disattivati, se un singolo aggiornamento può influire su più record del server e se i record del dataset client devono essere aggiornati all'atto dell'applicazione degli aggiornamenti. La proprietà *Options* è identica alla proprietà *Options* del provider. Pertanto, essa permette di impostare anche opzioni che non sono pertinenti o appropriate. Ad esempio, non c'è alcun motivo di includere *poIncFieldProps*, in quanto il dataset client non recuperara i propri dati da un dataset con campi persistenti. Per contro, non si vorrà escludere *poAllowCommandText*, incluso per default, perché ciò disattiverebbe la proprietà *CommandText* utilizzata da dataset client per specificare i dati da ottenere. Per informazioni sulla proprietà *Options* del provider, consultare il paragrafo "Impostazione di opzioni che influenzano i pacchetti dati" a pagina 28-5.

2 **Visualizzare e modificare i dati**, permettere l'inserimento di nuovi record e supportare la cancellazione di record esistenti. Sia la copia originale di ogni record

che tutte le modifiche vengono conservate in memoria. Questo procedimento è descritto in "Editing di dati" a pagina 27-5.

3 **Prelevare altri record in base alle necessità**. Per impostazione predefinita, i dataset client prelevano tutti i record e li conservano in memoria. Se un dataset contiene molti record o record con grandi campi BLOB, potrebbe essere opportuno modificare questo comportamento in modo che il dataset client prelevi solo la quantità necessaria di record da visualizzare e ne prelevi successivamente altri se necessario. Per i dettagli su come controllare il processo di prelievo dei record, vedere "Richiesta di dati da un dataset sorgente o da un documento" a pagina 27-28.

4 **A propria scelta, aggiornare i record**. Man mano che il tempo trascorre, altri utenti potrebbero modificare i dati sul server di database. Ciò fa sì che i dati del dataset client siano sempre più differenti dai dati sul server, accrescendo la possibilità che si verifichino errori al momento dell'applicazione degli aggiornamenti. Per attenuare questo problema, è possibile aggiornare i record che non sono stati ancora modificati. Per i dettagli, vedere "Rinnovo dei record" a pagina 27-33.

5 **Applicare i record nella cache locale al database o annullare gli aggiornamenti**. Per ogni record scritto nel database, viene innescato un evento *BeforeUpdateRecord*. Se si verifica un errore nel momento in cui viene scritto nel database un singolo record, un evento *OnUpdateError* permette all'applicazione di correggere l'errore, se possibile, e di continuare l'aggiornamento. Una volta completati gli aggiornamenti, tutti gli aggiornamenti applicati con successo vengono rimossi dalla cache locale. Per ulteriori informazioni sull'applicazione degli aggiornamenti al database, vedere "Aggiornamento dei record" a pagina 27-21.

   Invece di applicare aggiornamenti, un'applicazione può annullare gli aggiornamenti, svuotando il registro delle modifiche senza scrivere le modifiche nel database. È possibile annullare gli aggiornamenti chiamando il metodo *CancelUpdates*. Tutti i record cancellati presenti nella cache vengono ripristinati, i record modificati assumono i valori originali e i record appena inseriti spariscono.

Scelta del tipo di dataset per gli aggiornamenti in cache

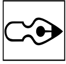
C++Builder include alcuni componenti dataset client specializzati per gli aggiornamenti in cache. Ogni dataset client è associato a un particolare meccanismo di accesso di dati. Essi vengono elencati nella Tabella Tabella 27.3:

Tabella 27.3 Dataset client specializzati per gli aggiornamenti in cache

Dataset client	Meccanismo di accesso ai dati
TBDEClientDataSet	Borland Database Engine
TSQLClientDataSet	dbExpress
TIBClientDataSet	InterBase Express

Inoltre, è possibile memorizzare in cache gli aggiornamenti utilizzando il dataset client generico (*TClientDataSet*) con un provider e un dataset sorgente esterni. Per

informazioni sull'uso di *TClientDataSet* con un provider esterno, consultare "Uso di un dataset client con un provider di dati" a pagina 27-26.



Anche i dataset client specializzati associati a un particolare meccanismo di accesso ai dati in effetti utilizzano un provider e un dataset sorgente. Tuttavia, sia il provider che il dataset sorgente sono interni al dataset client.

È più semplice utilizzare uno dei dataset client specializzati per gli aggiornamenti in cache. A volte, tuttavia, è preferibile utilizzare *TClientDataSet* con un provider esterno:

- Se si utilizza un meccanismo di accesso di dati che non ha un dataset client specializzato, è necessario utilizzare *TClientDataSet* con un componente provider esterno. Ad esempio, se i dati provengono da un documento XML o da un dataset custom.
- Se si opera con tabelle collegate mediante una relazione master/detail, si dovrebbe utilizzare *TClientDataSet* e connetterlo, utilizzando un provider, alla tabella master dei due dataset sorgente collegati con una relazione master/detail. Il dataset client vede il dataset di dettaglio come un campo di dataset annidato. Questa modalità operativa è necessaria per poter applicare gli aggiornamenti alle tabelle master e detail nell'ordine corretto.
- Se si vogliono creare gestori di evento che rispondono alla comunicazione fra il dataset client e il provider (ad esempio, prima e dopo che il dataset client ha prelevato i record dal provider), è necessario utilizzare *TClientDataSet* con un componente provider esterno. I dataset client specializzati pubblicano gli eventi più importanti per l'applicazione degli aggiornamenti (*OnReconcileError*, *BeforeUpdateRecord* e *OnGetTableName*), ma non pubblicano gli eventi inerenti alla comunicazione fra il dataset client e il suo provider, in quanto sono destinati principalmente alle applicazioni multi-tier.
- Se si utilizza BDE, potrebbe essere opportuno utilizzare un provider e un dataset sorgente esterni perché potrebbe essere necessario utilizzare un oggetto update. Benché sia possibile codificare un oggetto update a partire dal gestore di evento *BeforeUpdateRecord* di *TBDEClientDataSet*, è molto più semplice assegnare semplicemente la proprietà *UpdateObject* del dataset sorgente. Per informazioni sull'utilizzo di oggetti update, vedere "Aggiornamento di dataset mediante oggetti update" a pagina 24-41.

Indicazione dei record modificati

Mentre l'utente modifica un dataset client, può essere utile fornire un riscontro sulle modiche apportate. In particolar modo è utile nel caso si voglia permettere all'utente di annullare determinate modifiche, ad esempio, spostandosi sulla modifica e facendo clic su un pulsante "Annulla.

Il metodo *UpdateStatus* e le proprietà *StatusFilter* sono utili quando si vuole fornire un riscontro degli aggiornamenti effettuati:

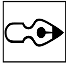
- *UpdateStatus* indica il tipo di aggiornamento, se vi è stato, che è stato apportato al record corrente. Può assumere uno dei seguenti valori:

- *usUnmodified* indica che il record corrente è invariato.
- *usModified* indica che il record corrente è stato modificato.
- *usInserted* indica un record che è stato inserito dall'utente.
- *usDeleted* indica un record che è stato cancellato dall'utente.

- *StatusFilter* controlla quali tipi di aggiornamento sono visibili nel registro delle modifiche. *StatusFilter* opera sui record nella cache in modo simile a quanto fanno i filtri sui normali dati. *StatusFilter* è un set, e pertanto può contenere una qualsiasi combinazione dei seguenti valori:

  - *usUnmodified* indica un record invariato.
  - *usModified* indica un record modificato.
  - *usInserted* indica un record inserito.
  - *usDeleted* indica un record cancellato.

  Per impostazione predefinita, *StatusFilter* è il set [*usModified*, *usInserted*, *usUnmodified*]. A questo set è possibile aggiungere *usDeleted* in modo da fornire un riscontro sui record cancellati.



UpdateStatus e *StatusFilter* sono utili anche nei gestori di evento *BeforeUpdateRecord* e *OnReconcileError*. Per informazioni su *BeforeUpdateRecord*, vedere "Interventi durante l'applicazione degli aggiornamenti" a pagina 27-23. Per informazioni su *OnReconcileError*, consultare "Riconciliazione degli errori di aggiornamento" a pagina 27-25.

Il seguente esempio mostra come fornire un riscontro sullo stato di aggiornamento dei record utilizzando il metodo *UpdateStatus*. L'esempio parte dall'assunto che la proprietà *StatusFilter* sia stata modificata includendo *usDeleted*, permettendo così la visualizzazione nel dataset dei record cancellati. Inoltre presuppone che al dataset sia stato aggiunto un campo calcolato di nome "Status."

```
void __fastcall TForm1::ClientDataSet1CalcFields(TDataSet *DataSet)
{
  switch (DataSet->UpdateStatus())
  {
    case usUnmodified:
      ClientDataSet1Status->Value = NULL; break;
    case usModified:
      ClientDataSet1Status->Value = "M"; break;
    case usInserted:
      ClientDataSet1Status->Value = "I"; break;
    case usDeleted:
      ClientDataSet1Status->Value = "D"; break;
  }
}
```

Aggiornamento dei record

Il contenuto del registro delle modifiche viene memorizzato come pacchetto dati nella proprietà *Delta* del dataset client. Per rendere permanenti le modifiche in *Delta*, il dataset client deve applicarle al database (o al dataset sorgente o al documento XML).

Quando un client applica gli aggiornamenti al server, si verifica quanto segue:

1 L'applicazione client chiama il metodo *ApplyUpdates* di un oggetto dataset client. Questo metodo passa il contenuto della proprietà *Delta* del dataset client al provider (interno o esterno). *Delta* è un pacchetto dati che contiene i record aggiornati, inseriti e cancellati del dataset client.

2 Il provider applica gli aggiornamenti, memorizzando nella cache tutti i record con problemi che non è in grado di risolvere da solo. Per informazioni dettagliate su come il provider applica gli aggiornamenti, consultare la sezione "Risposta alle richieste di aggiornamento del client" a pagina 28-8.

3 Il provider restituisce tutti i record non risolti al client dataset in un pacchetto dati *Result*. Il pacchetto dati *Result* contiene tutti i record che non sono stati aggiornati. Contiene inoltre informazioni sugli errori, come i messaggi e i codici di errore.

4 L'applicazione client dataset tenta di riconciliare gli errori di aggiornamento restituiti nel pacchetto *Result* record per record.

Applicazione degli aggiornamenti

Le modifiche apportate alla copia locale di dati del dataset client non vengono inviate al server di database (o al documento XML) finché l'applicazione client non chiama il metodo *ApplyUpdates* del dataset. *ApplyUpdates* prende le modifiche nel registro delle modifiche e le invia come pacchetto dati (denominato *Delta*) al provider. (Si noti che, nella maggior parte dei dataset client, il provider è interno al dataset client.)

ApplyUpdates accetta un singolo parametro, *MaxErrors*, che indica il numero massimo di errori che il provider può tollerare prima che venga interrotto l'intero processo di aggiornamento. Se *MaxErrors* è *0*, non appena si verifica un errore di aggiornamento, viene terminato l'intero processo di aggiornamento. Nessuna modifica viene scritta nel database e il registro delle modifiche del dataset client rimane intatto. Se *MaxErrors* è *-1*, viene tollerato un numero qualunque di errori e il registro delle modifiche contiene tutti i record che non è stato possibile aggiornare. Se *MaxErrors* è un valore positivo e si verifica un numero di errori maggiore di quello consentito da *MaxErrors*, tutti gli aggiornamenti vengono abbandonati. Se si verifica un numero di errori minore di quello specificato da *MaxErrors*, tutti i record aggiornati vengono cancellati automaticamente dal registro delle modifiche del dataset client.

ApplyUpdates restituisce il numero di errori effettivi incontrati, che è sempre minore o uguale a *MaxErrors* più uno. Questo valore di ritorno indica il numero di record che non possono essere scritti nel database.

Il metodo *ApplyUpdates* del dataset client compie le seguenti operazioni:

1 Chiama indirettamente il metodo *ApplyUpdates* del provider.Il metodo *ApplyUpdates* del provider scrive gli aggiornamenti nel database, nel dataset sorgente o nel documento XML e tenta di correggere tutti gli errori che incontra. I record che non è possibile aggiornare a causa di condizioni di errore vengono rinviati al dataset client.

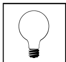
2 Il metodo *ApplyUpdates* del dataset client tenta quindi di riconciliare questi record con problemi chiamando il metodo *Reconcile*. *Reconcile* è una routine di gestione degli errori che chiama il gestore di evento *OnReconcileError*. Per correggere gli

errori è necessario scrivere il codice del gestore di evento *OnReconcileError*. Per informazioni su *OnReconcileError*, consultare "Riconciliazione degli errori di aggiornamento" a pagina 27-25.

3 Infine, *Reconcile* rimuove dal registro le modifiche apportate con esito positivo e aggiorna *Data* in modo da riflettere i record appena aggiornati. Quando *Reconcile* termina, *ApplyUpdates* riporta il numero di errori che si sono verificati.



In alcuni casi, il provider non è in grado di determinare come applicare gli aggiornamenti (ad esempio, quando applica gli aggiornamenti da una procedura registrata o da una join su più tabelle). I dataset e i componenti provider client generano eventi che permettono di gestire queste situazioni. Pe ri dettagli, vedere "Interventi durante l'applicazione degli aggiornamenti".



Se il provider è su un Application server senza stato, potrebbe essere opportuno cumunicare con lui a proposito delle informazioni di stato persistenti prima o dopo l'applicazione degli aggiornamenti. *TClientDataSet* riceve un evento *BeforeApplyUpdates* prima che vengano inviati gli aggiornamenti, il che consente di inviare al server le informazioni di stato persistenti. Dopo che gli aggiornamenti sono stati applicati (ma prima del processo di riconciliazione), *TClientDataSet* riceve un evento AfterApplyUpdatesin cui si può rispondere a tutte le informazioni di stato persistenti restituite dall'application server.

Interventi durante l'applicazione degli aggiornamenti

Quando un dataset client applica gli aggiornamenti, il provider determina come gestire la scrittura nel server di database o nel dataset sorgente di inserimenti, cancellazioni e modifiche. Quando si utilizza *TClientDataSet* con un componente provider esterno, si possono utilizzare le proprietà e gli eventi di quel provider per influire sul modo in cui vengono applicati gli aggiornamenti. Tali proprietà ed eventi sono descritti in "Risposta alle richieste di aggiornamento del client" a pagina 28-8.

Quando il provider è interno, tuttavia, come accade per qualsiasi dataset client associato a un meccanismo di accesso ai dati, è impossibile impostarne le proprietà o preparare gestori di evento. Di conseguenza il dataset client pubblica una proprietà e due eventi che permettono di influire sul modo in cui il provider interno applica aggiornamenti.

- *UpdateMode* controlla quali sono i campi utilizzati per individuare i record nelle istruzioni SQL generate dal provider per applicare gli aggiornamenti. *UpdateMode* è identico alla proprietà *UpdateMode* del provider. Per informazioni sulla proprietà *UpdateMode* del provider, consultare "Come influenzare la modalità di applicazione degli aggiornamenti" a pagina 28-10.
- *OnGetTableName* permette di fornire al provider il nome della tabella di database a cui dovrebbe applicare gli aggiornamenti. Questo permette al provider di generare le istruzioni SQL per gli aggiornamenti nel caso non sia in grado di identificare la tabella di database in base alla query specificata mediante *CommandText*. Ad esempio, se la query esegue una join su più tabelle che richiede l'aggiornamento di una singola tabella, la fornitura di un gestore di evento OnGetTableNamepermette al provider interno di applicare gli aggiornamenti in modo corretto.

Un gestore di evento *OnGetTableName* ha tre parametri: il componente provider interno, il dataset interno che ha prelevato i dati dal server e un parametro per restituire il nome della tabella da utilizzare nell'istruzione SQL generata.

- *BeforeUpdateRecord* si verifica per ogni record nel pacchetto delta. Questo evento permette di fare tutte le modifiche dell'ultimo minuto prima che il record sia inserito, cancellato o modificato. Inoltre fornisce al programmatore la possibilità di eseguire proprie istruzioni SQL per applicare l'aggiornamento nel caso in cui il provider non sia in grado di generare la corretta istruzione SQL (ad esempio, per join su più tabelle in cui si devono aggiornare più tabelle.)

  Un gestore di evento *BeforeUpdateRecord* ha cinque parametri: il componente provider interno, il dataset interno che ha prelevato i dati dal server, un pacchetto delta che è posizionato sul record che sta per essere aggiornato, un'indicazione se l'aggiornamento è un inserimento, una cancellazione o una modifica, e infine un parametro che restistuisce se il gestore di evento ha eseguito o meno l'aggiornamento. L'utilizzo è illustrato nel seguente gestore di evento. Per semplicità, l'esempio assume che le istruzioni SQL siano disponibili come globali variabili che necessitano solo di valori di campo:

```
void __fastcall TForm1::SQLClientDataSet1BeforeUpdateRecord(TObject *Sender,
   TDataSet *SourceDS, TCustomClientDataSet *DeltaDS, TUpdateKind UpdateKind, bool &Applied)
{
  TSQLConnection *pConn := (dynamic_cast<TCustomSQLDataSet *>(SourceDS)->SQLConnection);
  char buffer[256];
  switch (UpdateKind)
  case ukModify:
    // 1st dataset: update Fields[1], use Fields[0] in where clause
    sprintf(buffer, UpdateStmt1, DeltaDS->Fields->Fields[1]->NewValue,
            DeltaDS->Fields->Fields[0]->OldValue);
    pConn->Execute(buffer, NULL, NULL);
    // 2nd dataset: update Fields[2], use Fields[3] in where clause
    sprintf(buffer, UpdateStmt2, DeltaDS->Fields->Fields[2]->NewValue,
            DeltaDS->Fields->Fields[3]->OldValue);
    pConn->Execute(buffer, NULL, NULL);
    break;
  case ukDelete:
    // 1st dataset: use Fields[0] in where clause
    sprintf(buffer, DeleteStmt1, DeltaDS->Fields->Fields[0]->OldValue);
    pConn->Execute(buffer, NULL, NULL);
    // 2nd dataset: use Fields[3] in where clause
    sprintf(buffer, DeleteStmt2, DeltaDS->Fields->Fields[3]->OldValue);
    pConn->Execute(buffer, NULL, NULL);
    break;
  case ukInsert:
    // 1st dataset: values in Fields[0] and Fields[1]
    sprintf(buffer, UpdateStmt1, DeltaDS->Fields->Fields[0]->NewValue,
            DeltaDS->Fields->Fields[1]->NewValue);
    pConn->Execute(buffer, NULL, NULL);
    // 2nd dataset: values in Fields[2] and Fields[3]
    sprintf(buffer, UpdateStmt2, DeltaDS->Fields->Fields[2]->NewValue,
            DeltaDS->Fields->Fields[3]->NewValue);
    pConn->Execute(buffer, NULL, NULL);
    break;
```

```
}
```

Riconciliazione degli errori di aggiornamento

ThereEsistono due eventi che permettono di gestire gli errori che si verificano durante il processo di aggiornamento:

- Durante il processo di aggiornamento, il provider interno genera un evento *OnUpdateError* ogni volta che incontra un aggiornamento che non è in grado di gestire. Se si risolve il problema in un gestore di evento *OnUpdateError*, l'errore non veiene conteggiato nel numero massimo di errori passato al metodo *ApplyUpdates*. Questo evento si verifica solo per i dataset client che utilizzano un provider interno. Se si utilizza *TClientDataSet*, si può utilizzare invece l'evento *OnUpdateError* del componente provider.
- Dopo che l'intera operazione di aggiornamento è stata portata a termine, il dataset client genera un evento *OnReconcileError* per ogni record che il provider non è stato in grado di applicare al server di database.

I gestori di evento *OnReconcileError* o *OnUpdateError* devono essere sempre definiti a livello di codice, anche solo per scartare i record restituiti che non è stato possibile riconciliare. I gestori di evento di questi due eventi funzionano allo stesso modo. Essi includono i seguenti parametri:

- *DataSet:* Un dataset client che contiene il record aggiornato che non ha potuto essere applicato. È possibile usare questi metodi del dataset client per ottenere informazioni sui record con problemi e per modificare i record in modo da correggere tutti i problemi.In particolare, si devono usare le proprietà *CurValue*, *OldValue* e *NewValue* dei campi del record attivo per determinare la causa del problema di aggiornamento. Comunque, nel gestore di evento non di dovranno chiamare tutti i metodi del dataset client che modificano il record corrente.
- *E:* Un oggetto rappresenta il problema verificatosi. È possibile usare questa eccezione per estrarre un messaggio di errore o per determinare la causa dell'errore di aggiornamento.
- *UpdateKind:* Il tipo di aggiornamento che ha generato l'errore. *UpdateKind* può essere *ukModify* (il problema si è verificato aggiornando un record esistente che è stato modificato), *ukInsert* (il problema si è verificato inserendo un nuovo record), o *ukDelete* (il problema si è verificato cancellando un record esistente).
- *Action:* Un parametro riferimento che permette di indicare quale azione intraprendere all'uscita dal gestore di evento. Nel gestore di evento questo parametro viene impostato per
  - Saltare questo record, lasciandolo nel registro delle modifiche. (rrSkip o raSkip)
  - Bloccare l'intera operazione di riconciliazione. (rrAbort o raAbort)
  - Incorporare la modifica che non è andata a buon fine nel corrispondente record dal server. (raMerge) L'operazione è possibile solo se il record del server non include nessuna modifica ai campi modificati nel record del dataset client.

- Sostituire l'attuale aggiornamento nel registro delle modifiche con il valore del record nel gestore di evento, che presumibilmente è stato corretto. (rrApply o raCorrect)
- Ignorare l'errore completamente. (rrIgnore) Questa possibilità esiste solo nel gestore di evento *OnUpdateError* ed è destinata al caso in cui il gestore di evento applica l'aggiornamento al server di database. Il record aggiornato viene rimosso dal registro delle modifiche e unito in *Data*, come se il provider avesse applicato l'aggiornamento.
- Annullare le modifiche nel dataset client relativamente a questo record, ritornando ai valori forniti originariamente. (raCancel) Questa possibilità esiste solo nel gestore di evento *OnReconcileError*.
- Rinnovare il valore corrente del record per farlo corrispondere al record sul server. (raRefresh) Questa possibilità esiste solo nel gestore di evento *OnReconcileError*.

Il codice seguente, mostra un gestore di evento *OnReconcileError* che usa la finestra di dialogo di riconciliazione degli errori, inclusa nella unit RecError presente nella directory dell'Object Repository. (Per utilizzare questa finestra di dialogo, includere nella unit del sorgente il file RecError.hpp.)

```
void __fastcall TForm1::ClientDataSetReconcileError(TCustomClientDataSet *DataSet,
    EReconcileError *E, TUpdateKind UpdateKind, TReconcileAction &Action)
{
  Action = HandleReconcileError(this, DataSet, UpdateKind, E);
}
```

Uso di un dataset client con un provider di dati

Un dataset client utilizza un provider per fornirlo di dati e applicare aggiornamenti quando

- Memorizza in cache gli aggiornamenti da un server di database o da un altro dataset.
- Rappresenta i dati in un documento XML.
- Memorizza i dati nella parte client di un'applicazione multi-tier.

Per qualsiasi dataset client diverso da *TClientDataSet*, questo provider è interno e pertanto non è direttamente accessibile dall'applicazione. Con *TClientDataSet*, il provider è un componente esterno che collega il dataset client con una sorgente di dati esterna.

Un componente provider esterno può risiedere nella stessa applicazione del dataset client oppure può fare parte di un'applicazione separata in esecuzione su un altro sistema. Per ulteriori informazioni sui componenti provider, consultare il Capitolo 28, "Uso di componenti provider". Per maggiori informazioni sulle applicazioni in cui il provider è in un'applicazione separata su un altro sistema, consultare il Capitolo 29, "Creazione di applicazioni multi-tier".

Quando si utilizza un provider (interno o esterno), il dataset client memorizza sempre in cache qualsiasi aggiornamento. Per le informazioni relative, consultare "Utilizzo di un dataset client per registrare in cache gli aggiornamenti" a pagina 27-17.

Le sezioni seguenti descrivono ulteriori proprietà e metodi del dataset client che gli permettono di operare con un provider.

Specifica di un provider

A differenza dei dataset client che sono associati a un meccanismo di accesso ai dati, *TClientDataSet* non ha alcun componente provider interno per assemblare i dati o applicare gli aggiornamenti. Se lo si vuole utilizzare per rappresentare i dati da un dataset sorgente o da un documento XML, sarà perciò necessario associare il dataset client a un componente provider esterno.

Il modo in cui si associa *TClientDataSet* a un provider dipende dal fatto che il provider sia nella stessa applicazione del dataset client o sia su un Application server remoto, in esecuzione su un altro sistema.

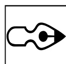
- Se il provider è nella stessa applicazione del dataset client, è possibile associarlo a un provider nell'Object Inspector scegliendo un provider dalla lista a discesa della proprietà *ProviderName*. Questa modalità funziona finché il provider ha lo stesso *Owner* del dataset client. (Il dataset client e il provider hanno lo stesso *Owner* se sono collocati sulla stessa scheda o sullo stesso modulo dati). Per utilizzare un provider locale che ha un *Owner* differente, si deve formare l'associazione in fase di esecuzione utilizzando il metodo *SetProvider* del dataset client.

  Se si prevede di passare all'utilizzo di un provider remoto o si desidera chiamare direttamente l'interfaccia *IAppServer*, è possibile anche impostare la proprietà *RemoteServer* a un componente *TLocalConnection*. Se si utilizza *TLocalConnection*, l'istanza di *TLocalConnection* gestisce la lista di tutti i provider locali all'applicazione e gestisce le chiamate a *IAppServer* del dataset client. Se non si utilizza *TLocalConnection*, l'applicazione crea un oggetto nascosto che gestisce le chiamate a *IAppServer* dal dataset client.

- Se il provider è su un application server remoto, quindi, oltre alla proprietà *ProviderName*, è necessario specificare un componente che connette il dataset client all'application server. Esistono due proprietà che possono gestire questa attività: *RemoteServer*, che specifica il nome di un componente connection da cui ottenere una lista di provider, oppure *ConnectionBroker*, che specifica un broker centralizzato che fornisce un ulteriore livello di indirezione fra il dataset client e il componente connection. Il componente connection e, se utilizzato, il broker di connessione, risiedono nello stesso modulo dati del dataset client. Il componente connection stabilisce e gestisce una connessione con un application server, detto anche "data broker". Per ulteriori informazioni, consultare "Struttura dell'applicazione client" a pagina 29-4.

  In progettazione, dopo aver specificato *RemoteServer* o *ConnectionBroker*, nell'Object Inspector si può selezionare un provider dall'elenco a discesa della proprietà *ProviderName*. Questa lista include sia i provider locali (nella stessa

scheda o modulo dati) sia i provider remoti a cui è possibile accedere attraverso il componente connection.



Se il componente connection è un'istanza di *TDCOMConnection*, l'application server deve essere registrato sulla macchina client.

In esecuzione, è possibile utilizzare uno dei fornitori disponibili (locali e remoti) impostando da programma la proprietà *ProviderName*.

Richiesta di dati da un dataset sorgente o da un documento

I dataset client possono controllare come prelevano i propri pacchetti dati da un provider. Per impostazione predefinita, essi recuperano tutti i record dal dataset sorgente. Questo è vero sia nel caso che il dataset sorgente e il provider siano componenti interni (come nel caso di *TBDEClientDataSet*, *TSQLClientDataSet* e *TIBClientDataSet*), sia nel caso che siano componenti separati che forniscono i dati a *TClientDataSet*.

È possibile cambiare il modo in cui il dataset client preleva i record utilizzando le proprietà *PacketRecords* e *FetchOnDemand*.

Recupero incrementale

Modificando la proprietà *PacketRecords*, è possibile specificare che il dataset client preleva i dati in lotti più piccoli. *PacketRecords* specifica quanti record recuperare per volta, oppure il tipo di record da restituire. Per impostazione predefinita, *PacketRecords* ha come valore *-1*, che indica che tutti i record disponibili vengono recuperati contemporaneamente quando il dataset client viene aperto per la prima volta, o quando l'applicazione chiama in modo esplicito *GetNextPacket*. Quando *PacketRecords* è *-1*, dopo che i dati sono stati recuperati la prima volta, non è mai necessario che un dataset client li recuperi ulterioriormente, in quanto ha già tutti i record disponibili.

Per recuperare i record in piccoli blocchi, basta impostare *PacketRecords* al numero di record da recuperare. Per esempio, l'istruzione seguente imposta le dimensioni di ciascun pacchetto dati a dieci record:

```
ClientDataSet1->PacketRecords = 10;
```

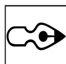
Questa procedura di recupero dei record a blocchi viene denominato "recupero incrementale". I dataset client utilizzano il recupero incrementale quando *PacketRecords* è maggiore di zero.

Per recuperare ogni lotto di record, il dataset client chiama *GetNextPacket*. I pacchetti appena recuperati vengono aggiunti alla fine dei dati già presenti nel dataset client. *GetNextPacket* restituisce il numero di record che recupera. Se il valore restituito è uguale a quello di *PacketRecords*, non viene incontrata la fine dei record disponibili. Se il valore restituito è maggiore di *0* ma minore di *PacketRecords*, durante l'operazione di recupero è stato raggiunto l'ultimo record. Se *GetNextPacket* restituisce *0*, non ci sono più record da recuperare.



Il recupero incrementale non è possibile se si stanno prelevando dati da un provider remoto su un Application server senza stato. Per informazioni su come utilizzare il

recupero incrementale in un modulo dati remoto senza stato, consultare la sezione “Supporto delle informazioni di stato nei moduli dati remoti” a pagina 29-20.



È anche possibile utilizzare *PacketRecords* per recuperare informazioni di metadati sul dataset sorgente. Per recuperare le informazioni dei metadati, occorre impostare *PacketRecords* a *0*.

Recupero in automatico

Il recupero automatico dei record viene controllato dalla proprietà *FetchOnDemand*. Se *FetchOnDemand* è **true** (impostazione predefinita), il dataset client recupera automaticamente i record man mano che occorrono. Per disattivare questa opzione, basta impostare *FetchOnDemand* a **false**. Quando *FetchOnDemand* è **false**, per recuperare i record l’applicazione deve chiamare in modo esplicito *GetNextPacket*.

Ad esempio, le applicazioni che devono rappresentare dataset a sola lettura molto vasti possono disattivare FetchOnDemandper avere la certezza che i dataset client non tentino di caricare più dati di quanti ce ne possono stare in memoria. Tra i diversi recuperi il dataset client libera la sua cache utilizzando il metodo *EmptyDataSet*. Questo approccio, tuttavia, non funziona bene quando il client deve inoltrare gli aggiornamenti al server.

Il provider controlla se i record nei pacchetti dati includono dati BLOB e se contengono dataset di dettaglio annidati. Se il provider esclude dai record queste informazioni, la proprietà *FetchOnDemand* fa sì che il dataset client prenda automaticamente i dati BLOB e i dataset di dettaglio man mano che servono. Se *FetchOnDemand* è **false** e il provider non include nei record dati BLOB o dataset di dettaglio, per ottenere queste informazioni è necessario chiamare esplicitamente il metodo *FetchBlobs* o *FetchDetails*.

Ottenimento dei parametri dal dataset sorgente

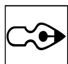
Esistono due circostanze in cui un dataset client ha bisogno di ottenere i valori dei parametri:

- L’applicazione ha la necessità di conoscere il valore dei parametri di output in una procedura registrata.
- L’applicazione vuole inizializzare i parametri di input di una query o di una procedura registrata con i valori correnti nel dataset sorgente.

I dataset client memorizzano valori dei parametri nella proprietà *Params*. Questi valori vengono aggiornati con i parametri di output ogni volta che il dataset client recupera i dati dal dataset sorgente. A volte, comunque, un componente *TClientDataSet* di un’applicazione client ha bisogno di acquisire i parametri di output senza dover recuperare dati.

Per recuperare i parametri di output, pur senza recuperare record, o per inizializzare i parametri di input, il client può richiedere i valori dei parametri dal dataset sorgente chiamando il metodo *FetchParams*. I parametri vengono restituiti in un pacchetto dati dal provider e assegnati alla proprietà *Params* del dataset client.

In fase di progettazione, la proprietà *Params* può essere inizializzata facendo clic con il pulsante destro sul dataset client e scegliendo Fetch *Params*.



Non è quasi mai necessario chiamare *FetchParams* quando il dataset client utilizza un provider interno e un dataset sorgente perché la proprietà *Params* rispecchia sempre i parametri del dataset sorgente interno. Con *TClientDataSet*, il metodo *FetchParams* (o il comando Fetch *Params*) funziona solo se il dataset client è connesso a un provider il cui dataset associato è in grado di fornire parametri. Ad esempio, se il dataset sorgente è dataset di tipo tabella non vi sono parametri da ottenere.

Se il provider è su un sistema separato, come facente parte di un Application server senza stato, non è possibile utilizzare *FetchParams* per acquisire i parametri di output. In un application server senza stato, altri client potrebbero modificare o rieseguire la query o la procedura registrata, modificando pertanto i parametri di output prima che avvenga la chiamata a *FetchParams*. Per recuperare i parametri di output da un application server senza stato, si deve utilizzare il metodo *Execute*. Se il provider è associato con una query o con una procedura registrata, *Execute* chiede al provider di eseguire la query o la procedura registrata e di restituire tutti i parametri di output. I parametri così restituiti vengono quindi utilizzati per aggiornare automaticamente la proprietà *Params*.

Passaggio di parametri al dataset sorgente

I dataset client possono passare dei parametri al dataset sorgente per specificare quali dati desiderano ricevere nei pacchetti dati che esso invia. Questi parametri possono specificare:

- I valori dei parametri per una query o una procedura registrata che viene eseguita sull’application server
- I valori di campi che limitano i record inviati nei pacchetti dati

È possibile specificare i valori dei parametri che il dataset client invia al dataset sorgente sia in progettazione sia in esecuzione. In progettazione, selezionare il dataset client e fare doppio clic nell’Object Inspector sulla proprietà *Params*. Ciò richiama il Collection editor, dove è possibile aggiungere, rimuovere o risistemare i parametri. Selezionando un parametro nel Collection editor, si può usare l’Object Inspector per modificarne le proprietà.

In esecuzione, per aggiungere parametri al dataset client, bisogna ricorre al metodo *CreateParam* della proprietà *Params*. *CreateParam* restituisce un oggetto parametro, dati un nome specifico, un tipo di parametro e un datatype. È quindi possibile utilizzare le proprietà di quell’oggetto parametro per assegnare un valore al parametro.

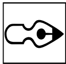
Per esempio, il codice seguente aggiunge un parametro di input di nome CustNo e con un valore di 605:

```
TParam *pParam = ClientDataSet1->Params->CreateParam(ftInteger, "CustNo", ptInput);
pParam->AsInteger = 605;
```

Se il dataset client non è attivo, si possono inviare i parametri all'application server e recuperare un pacchetto dati che rifletta quei valori di parametro semplicemente impostando la proprietà *Active* a **true**.

Invio dei parametri di una query o di una procedura registrata

Quando la proprietà *CommandType* del dataset client è *ctQuery* o *ctStoredProc*, oppure, nel caso il dataset client sia un'istanza di *TClientDataSet*, quando il provider associato rappresenta il risultato di una query o di una procedura registrata, è possibile utilizzare la proprietà Paramsper specificare i valori dei parametri. Quando il dataset client richiede i dati dal dataset sorgente, o utilizza il suo metodo *Execute* per eseguire una query o una procedura registrata che non restituisce un dataset, passa questi valori di parametro insieme alla richiesta di dati o al comando da eseguire. Quando il provider riceve questi valori del parametro, li assegna al proprio dataset associato. Esso quindi incarica il dataset di eseguire la propria query o procedura registrata utilizzando questi valori di parametro, e, nel caso il dataset client abbia richiesto dei dati, inizia a fornirglieli, iniziando dal primo record nel set risultato.



I nomi dei parametri devono coincidere con quelli utilizzati nel dataset sorgente.

Limitazione dei record mediante parametri

Se il dataset client è

- un'istanza di *TClientDataSet* il cui provider associato rappresenta un componente *TTable* o *TSQLTable*, oppure
- un'istanza di *TSQLClientDataSet* o di *TBDEClientDataSet* la cui proprietà *CommandType* è *ctTable*

allora può utilizzare la proprietà *Params* per limitare il numero di record che memorizza nella cache in memoria. Ogni parametro rappresenta un valore di campo che deve trovare una corrispondenza prima che il record possa essere incluso nei dati del dataset client. Questo funziona come un filtro, salvo che con un filtro, i record sono ancora memorizzati nella cache in memoria, ma non sono disponibili.

Ogni nome di parametro deve coincidere con il nome di un campo. Se si utilizza *TClientDataSet*, questi sono i nomi dei campi nel componente *TTable* o *TSQLTable* associato al provider. Se si utilizza *TSQLClientDataSet* o *TBDEClientDataSet*, questi sono i nomi dei campi nella tabella sul server di database.I dati nel dataset client allora includono solo quei i record i cui valori nei campi corrispondenti coincidono con i valori assegnati ai parametri.

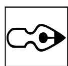
Si consideri, per esempio, un'applicazione che visualizza gli ordini di un unico cliente. Quando l'utente identifica il cliente, il dataset client imposta la sua proprietà *Params* includendo un unico parametro di nome CustID (o con il nome con cui viene chiamato il campo nella tabella della sorgente), il cui valore identifica il cliente di cui verranno visualizzati gli ordini. Nel momento in cui il dataset client richiede i dati della sorgente dataset, passa questo valore di parametro. Il provider invia, quindi, solo i record relativi al cliente identificato. Ciò risulta molto più efficiente rispetto al consentire al provider di inviare all'applicazione client tutti i record degli ordini e successivamente filtrare i record utilizzando il dataset client.

Gestione dei vincoli dal server

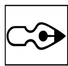
Se un server di database definisce dei vincoli sulla validità dei dati, è bene che il dataset sia consapevole della loro esistenza. In questo modo, il dataset client può fare in modo che le modifiche dell'utente non violino mai i vincoli del server.In questo modo, tali violaziono non saranno mai passate al server di database dove sarebbero rifiutate. Ciò significa che vi sarà un minor numero di aggiornamenti che genereranno condizioni di errore durante il processo di aggiornamento.

Indipendentemente dalla sorgente dati, è possibile duplicare quei vincoli del server aggiungendoli esplicitamente al dataset client. Questo procedimento è descritto in "Specifica di vincoli custom" a pagina 27-8.

È più comodo, comunque, che i vincoli del server siano inclusi automaticamente nei pacchetti dati. In questo modo non è necessario specificare esplicitamente le espressioni e i vincoli predefiniti e il dataset client modificherà i valori sotto controllo quando cambiano i vincoli server. Per default, questo è esattamente ciò che accade: se il dataset sorgente è a conoscenza dei vincoli del server, il provider li includerà automaticamente nei pacchetti dati e il dataset client li imporrà nel momento in cui l'utente registra le modifiche nel registro delle modifiche.



Solo i dataset che utilizzano BDE possono importare i vincoli dal server. Ciò significa che i vincoli del server sono inclusi nei pacchetti dati solo se si utilizza *TBDEClientDataSet* o *TClientDataSet* con un provider che rappresenta un dataset basato su BDE. Per ulteriori informazioni su come importare i vincoli del server e su come impedire a un provider di includerli nei pacchetti dati, vedere "Gestione dei vincoli del server" a pagina 28-13.



Per maggiori informazioni sulle operazioni con i vincoli, una volta che sono stati importati, consultare la sezione "Uso dei vincoli del server" a pagina 23-23.

Anche se l'importazione dei vincoli e delle espressioni del server è una funzionalità estremamente preziosa che consente a un'applicazione di salvaguardare l'integrità dei dati, a volte potrebbe essere necessario disattivare temporaneamente i vincoli. Ad esempio, se un vincolo del server si basa sul valore massimo corrente di un campo, ma il dataset client utilizza il recupero incrementale, potrebbe accadere che il valore massimo corrente di quel campo nel dataset client sia diverso dal valore massimo sul server di database, e i vincoli potrebbero essere richiamati diversamente. Per fare un altro esempio, nel caso un dataset client applichi ai record un filtro mentre i vincoli sono abilitati, il filtro potrebbe interferire in modo involontario con le condizioni di vincolo. In ognuno di questi casi, un'applicazione potrebbe disattivare il controllo dei vincoli.

Per disattivare temporaneamente i vincoli, chiamare il metodo *DisableConstraints*. Ogni volta che si chiama *DisableConstraints*, viene incrementato un contatore dei riferimenti. Finché il contatore dei riferimenti è maggiore di zero, i vincoli non vengono imposti sul dataset client.

Per riattivare i vincoli sul dataset client, si deve chiamare il metodo *EnableConstraints* del dataset. Ogni chiamata a *EnableConstraints* decrementa il contatore dei riferimenti. Quando il contatore dei riferimenti è zero, i vincoli risultano nuovamente abilitati.



DisableConstraints e *EnableConstraints* devono essere chiamati sempre in coppia per essere certi che i vincoli possano essere attivati quando occorre.

Rinnovo dei record

Le applicazioni client lavorano con un'istantanea in memoria dei dati del dataset sorgente. Se il dataset sorgente rappresenta i dati di un server, man mano che il tempo trascorre, tali dati potrebbero essere modificati da altri utenti. I dati nel dataset client costituiscono una rappresentazione sempre meno precisa dei dati originali.

Come tutti i dataset, i dataset client hanno un metodo *Refresh* che rinnova i propri record per farli corrispondere ai valori correnti sul server. Tuttavia, la chiamata a *Refresh* funziona solo se nel registro delle modifiche non c'è alcuna modifica. La chiamata a *Refresh* quando vi sono modifiche non applicate provoca un'eccezione.

I dataset client possono anche aggiornare i dati lasciando inalterato il registro delle modifiche. Per fare ciò, chiamare il metodo *RefreshRecord*. A differenza del metodo *Refresh*, *RefreshRecord* rinnova solo il record corrente del dataset client. *RefreshRecord* modifica il valore del record ottenuto originariamente dal provider, ma lascia le modifiche nel registro delle modifiche.



Non è detto che sia sempre opportuno chiamare *RefreshRecord*.Se le modifiche dell'utente entrano in conflitto con quelle apportate al dataset sottostante da altri utenti, la chiamata a *RefreshRecord* maschererà questo conflitto. Quando il dataset client applicherà i suoi aggiornamenti, non si verificheranno errori di riconciliazione e l'applicazione non potrà risolvere il conflitto.

Per evitare di mascherare gli errori di aggiornamento, prima di chiamare *RefreshRecord* potrebbe essere opportuno controllare che non che non vi siano aggiornamenti pendenti. Ad esempio, il seguente *AfterScroll* aggiorna il record corrente ogni volta che l'utente passa su un nuovo record (garantendo il valore più recente), ma solo se è possibile farlo in tutta sicurezza.

```
void __fastcall TForm1::ClientDataSet1AfterScroll(TDataSet *DataSet)
{
  if (ClientDataSet1->UpdateStatus == usUnModified)
    ClientDataSet1->RefreshRecord();
}
```

Comunicazione con i provider tramite eventi custom

I dataset client comunicano con un componente provider tramite un'interfaccia speciale di nome *IAppServer*. Se il provider è locale, *IAppServer* è l'interfaccia verso un oggetto generato automaticamente che gestisce tutta le comunicazioni fra il dataset client e il proprio provider. Se il provider è remoto, *IAppServer* è l'interfaccia di un oggetto COM associato di un modulo dati remoto sull'Application server, oppure (nel caso di un server SOAP) è un'interfaccia generata dal component connection.

TClientDataSet offre diverse opportunità per personalizzare la comunicazione che usa l'interfaccia *IAppServer*. Prima e dopo ogni chiamata al metodo *IAppServer* indirizzata al provider del dataset client, *TClientDataSet* riceve speciali eventi che

consentono di comunicare al suo provider le informazioni più disparate. Tali eventi sono abbinati ad analoghi eventi sul provider. Così ad esempio, quando il dataset client chiama il proprio metodo *ApplyUpdates*, si verificano gli eventi seguenti:

1. Il dataset client riceve un evento *BeforeApplyUpdates*, in cui specifica informazioni personalizzate arbitrarie utilizzando un OleVariant di nome *OwnerData*.
2. Il provider riceve un evento *BeforeApplyUpdates*, in cui può rispondere alle informazioni in *OwnerData* pervenute dal dataset client e aggiornare il valore di *OwnerData* con nuove informazioni.
3. Il provider prosegue il normale processo di assemblaggio del pacchetto dati (inclusi tutti gli eventi abbinati).
4. Il provider riceve un evento *AfterApplyUpdates* in cui può rispondere al valore corrente di *OwnerData* e aggiornarlo con un valore da passare al dataset client.
5. Il dataset client riceve un evento *AfterApplyUpdates*, in cui può rispondere al valore di *OwnerData* restituito.

Tutte le altre chiamate ai metodi di *IAppServer* sono abbinate a set analoghi di eventi *BeforeXXX* e *AfterXXX* che consentono di personalizzare la comunicazione tra il dataset client e il provider.

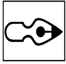
Inoltre, il dataset client possiede uno speciale metodo *DataRequest* il cui unico scopo è quello di consentire una comunicazione con il provider, specifica da applicazione ad applicazione. Quando il dataset client chiama *DataRequest*, passa come parametro un OleVariant che contiene una qualsiasi informazione. A sua volta, ciò genera sul provider un evento *OnDataRequest*, in cui è possibile rispondere in uno dei modi previsti dall'applicazione e restituire un valore al dataset client.

Ridefinizione del dataset sorgente

I dataset client associati a un particolare meccanismo di accesso ai dati utilizzano le proprietà *CommandText* e *CommandType* per specificare i dati che essi rappresentano. Se si utilizza *TClientDataSet*, tuttavia, i dati sono specificati dal dataset sorgente, non dal dataset client. Di solito, questo dataset sorgente ha una proprietà che specifica un'istruzione SQL per generare i dati, oppure il nome di una tabella di database o di una procedura registrata.

Se il provider lo consente, *TClientDataSet* può ridefinire la proprietà sul dataset sorgente che indica quali dati esso rappresenta. Cioè, se il provider lo consente, la proprietà *CommandText* del dataset client sostituisce la proprietà che specifica quali dati rappresenta sul dataset del provider. Ciò consente a *TClientDataSet* di specificare dinamicamente quali sono dati vuole vedere.

Per impostazione predefinita, i componenti provider esterni non permettono ai dataset client di utilizzare il valore di *CommandText* in questo modo. Per consentire a un *TClientDataSet* di utilizzare la sua proprietà *CommandText*, è necessario aggiungere *poAllowCommandText* alla proprietà *Options* del provider. In caso contrario, il valore di *CommandText* sarà ignorato.



Non rimuovere *poAllowCommandText* dalla proprietà Optionsdi *TSQLClientDataSet*, *TBDEClientDataSet* o *TIBClientDataSet*. La proprietà *Options* del dataset client viene inviata al provider interno, e pertanto la rimozione di *poAllowCommandText* impedisce al dataset client di specificare a quali dati accedere.

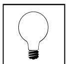
Il dataset client invia al provider la stringa *CommandText* in due occasioni:

- Quando il dataset client si apre la prima volta. Dopo aver recuperato il primo pacchetto dati del provider, il dataset client non invia più *CommandText* mentre recupera i successivi.
- Quando il dataset client invia un comando *Execute* al provider.

Per inviare un comando SQL o modificare il nome della tabella o della procedura registrata in un qualsiasi momento successivo, si deve usare esplicitamente l'interfaccia *IAppServer* che è disponibile come proprietà *IAppServer*. Questa proprietà rappresenta l'interfaccia attraverso cui il dataset client comunica con il proprio provider.

Uso di un dataset client con dati basati su file

I dataset client possono operare con file dedicati su disco oltre che con dati di un server. Ciò consente di utilizzarli in applicazioni database basate su file o che utilizzano il "modello briefcase". I file speciali che i dataset client utilizzano per i propri dati sono chiamati MyBase.



Tutti i dataset client sono utilizzabili nelle applicazioni che seguono il modello briefcase, ma per un'applicazione MyBase pura (un'applicazione che non utilizza un provider), è preferibile utilizzare *TClientDataSet*, in quanto implica un minor costo gestionale.

In un'applicazione MyBase, l'applicazione client non può ottenere le definizioni di tabella e i dati dal server, e non esiste alcun server a cui può applicare gli aggiornamenti. Invece, in modo del tutto indipendente, il dataset client deve:

- Definire e creare le tabelle
- Caricare i dati salvati
- Fondere le modifiche nei dati
- Salvare i dati

Creazione di un nuovo dataset

Esistono tre modi per definire e per creare dei dataset client che non rappresentano dati di un server:

- È possibile definire e creare un nuovo dataset client utilizzando i campi persistenti o le definizioni dei campi e degli indici. Questo segue lo stesso schema della creazione di un qualsiasi dataset di tipo tabella. Per ulteriori informazioni consultare "Creazione e cancellazione di tabelle" a pagina 22-40.

- È possibile copiare un dataset esistente (in progettazione o in esecuzione). Per maggiori informazioni sulla copia di dataset esistenti, consultare il paragrafo “Copia di dati da un altro dataset” a pagina 27-15.
- È possibile creare un dataset client da un qualsiasi documento XML. Per i dettagli, vedere “Conversione di documenti XML in pacchetti dati” a pagina 30-6.

Una volta creati il dataset, è possibile salvarlo in un file. Da questo momento in poi, non sarà più necessario ricreare la tabella, ci si limiterà a caricarla dal file salvato. Quando si inizia un’applicazione database basata su file, prima di iniziare a scrivere l’applicazione stessa, sarebbe oppurtuno per prima cosa creare e salvare i file vuoti per i dataset. In questo modo, si inizia con i metadati per il dataset client già definito, semplificando così la configurazione dell’interfaccia utente.

Caricamento dei dati da un file o da uno stream

Per caricare i dati da un file, si deve chiamare il metodo *LoadFromFile* di un dataset client. *LoadFromFile* richiede un solo parametro, una stringa che specifica il file da cui leggere i dati. Il nome del file deve essere, se occorre, completo di percorso. Se si caricano sempre i dati del dataset client dallo stesso file, si può invece utilizzare la proprietà *FileName*. Se *FileName* contiene un file esistente, i dati vengono caricati automaticamente non appena viene aperto il dataset client.

Per caricare i dati da uno stream, si deve chiamare il metodo *LoadFromStream* del dataset client. *LoadFromStream* richiede un solo parametro, un oggetto stream che fornisce i dati.

I dati caricati tramite *LoadFromFile* (*LoadFromStream)* devono essere stati precedentemente salvati utilizzando un formato dati del dataset client da questo o da un altro dataset client con il metodo *SaveToFile* (*SaveToStream*), opppure generati a parire da un documento XML. Per ulteriori informazioni sul salvataggio dei dati in un file o in uno stream, consultare il paragrafo “Salvataggio dei dati in un file o in uno stream” a pagina 27-37. Per informazioni sulla creazione di dati di dataset client a partire da un documento XML, vedere il Capitolo 30, “Uso di XML nelle applicazioni database”.

Quando si chiama *LoadFromFile* o *LoadFromStream*, tutti i dati del file vengono letti nella proprietà *Data*. Tutte le modifiche che erano presenti nel relativo registro quando i dati sono stati salvati vengono letti nella proprietà *Delta*. Tuttavia, gli unici indici che vengono letti dal file sono quelli che sono stati creati con il dataset.

Fusione delle modifiche nei dati

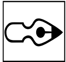
Quando si apportano modifiche a un dataset client, tutte le modifiche ai dati esistono solo in un registro delle modifiche in memoria. Questo file di registrazione può essere gestito separatamente dai dati stessi, benché sia completamente trasparente agli oggetti che utilizzano il dataset client. Vale a dire, ad esempio, che i controlli che consentono di spostarsi nel dataset client o ne visualizzano i dati vedono una vista dei dati che include le modifiche. Se non si desidera annullare le modifiche, tuttavia, si dovrebbe unire il registro delle modifiche nei dati del dataset client chiamando il

metodo *MergeChangeLog*. *MergeChangeLog* sovrascrive i record in *Data* con i valori di campo modificati presenti nel registro delle modifiche.

Dopo l'esecuzione di *MergeChangeLog*, *Data* contiene un misto di dati esistenti e delle modifiche che si trovavano nel registro. Questa miscela diventa la nuova struttura di *Data* alla quale possono essere apportate ulteriori modifiche. *MergeChangeLog* rimuove dal registro delle modifiche tutti i record e riporta la proprietà *ChangeCount* a *0*.



Non chiamare *MergeChangeLog* per i dataset client che utilizano un provider. In questo caso, per scrivere le modifiche nel database si deve chiamare *ApplyUpdates*. Per ulteriori informazioni, consultare "Applicazione degli aggiornamenti" a pagina 27-22.



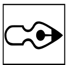
È anche possibile fondere le modifiche nei dati di un dataset client diverso se questo originariamente ha fornito i dati nella proprietà *Data*. Per ottenere questo risultato, occorre utilizzare un dataset provider. Per un esempio su come farlo, consultare "Assegnazione diretta dei dati" a pagina 27-15.

Se non si vogliono utilizzare le funzionalità estese di annullamento del registro delle modifiche, è possibile impostare la proprietà *LogChanges* del dataset client a **false**. Quando *LogChanges* è **false**, le modifiche vengono combinate automaticamente quando si registrano i record e non c'è necessità di chiamare *MergeChangeLog*.

Salvataggio dei dati in un file o in uno stream

Anche dopo aver unito le modifiche nei dati del dataset client, questi dati esistono ancora solo in memoria. Benché siano persistenti, se si chiude il dataset client e lo si riapre nell'applicazione, tali dati scompariranno nel momento in cui si chiude l'applicazione. Per rendere i dati permanenti, devono essere scritti sul disco. Scrivere le modifiche sul disco utilizzando il metodo *SaveToFile*.

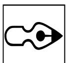
SaveToFile richiede un solo parametro, una stringa che specifica il file in cui scrivere i dati. Il nome del file deve essere, se occorre, completo di percorso. Se il file esiste già, il suo contenuto viene completamente sovrascritto.



SaveToFile non salvaguarda gli indici eventualmente aggiunti al dataset client in esecuzione, ma solo gli indici aggiunti al momento della creazione del dataset client.

Se si salvano sempre i dati nello stesso file, si può invece ricorre alla proprietà *FileName*. Se *FileName* è impostata, i dati vengono salvati automaticamente nel file indicato quando il dataset client viene chiuso.

È anche possibile salvare i dati in uno stream utilizzando il metodo *SaveToStream*. *SaveToStream* richiede un solo parametro, un oggetto stream che riceve i dati.



Se si salva un dataset client mentre ci sono ancora modifiche nel relativo registro, queste non vengono fuse con i dati. Quando si ricaricano i dati, utilizzando il metodo *LoadFromFile* o *LoadFromStream*, il registro delle modifiche conterrà ancora le modifiche che non sono state fuse. Ciò è importante per le applicazioni che sopportano il modello briefcase, nel quale quelle modifiche devono magari essere applicate a un componente provider nell'application server.

Capitolo

28

Uso di componenti provider

I componenti provider (*TDataSetProvider* e *TXMLTransformProvider*) forniscono il meccanismo in base al quale i dataset client ottengono i loro dati. I provider

- Ricevono le richieste di dati da un dataset client (o da un broker XML), prelevano i dati richiesti, assemblano i dati in un pacchetto dati trasportabile e restituiscono i dati al dataset client (o al borker XML). Questa attività viene definita "fornitura."
- Ricevono i dati aggiornati da un dataset client (o da un broker XML), applicano gli aggiornamenti al server di database o al dataset sorgente o al documento sorgente XML e registrano tutti gli aggiornamenti che non possono essere applicati, restituendo gli aggiornamenti insoluti al dataset client per una ulteriore riconciliazione. Questa attività viene definita "risoluzione."

Quasi tutto il lavoro di un componente provider viene svolto automaticamente. Non è necessario scrivere alcun programma sul provider per creare i pacchetti dati a partire dai dati in un dataset o in un documento XML o per applicare gli aggiornamenti. Tuttavia, i componenti provider includono un numero di eventi e proprietà che consentono all'applicazione un controllo più diretto su quali informazioni inserire nei pacchetti da inviare al client e sulla modalità di risposta dell'applicazione alle richieste del client.

Quando si utilizza *TBDEClientDataSet*, *TSQLClientDataSet* o *TIBClientDataSet*, il provider è interno al dataset client e l'applicazione non ha alcun accesso diretto a esso. Quando si utilizza *TClientDataSet* o *TXMLBroker*, invece, il provider è un componente separato che è possibile utilizzare per controllare quali informazioni vengono assemblate in pacchetti per i client e per rispondere agli eventi che si verificano durante il processo di fornitura e di risoluzione. I dataset client che hanno un provider interno fanno emergere alcune proprietà ed eventi del provider interno come se fossero proprietà ed eventi propri del dataset, ma per un maggiore controllo, si potrebbe preferire l'utilizzo di *TClientDataSet* con un componente provider separato.

Quando si utilizza un componente provider separato, tale componente può risiedere nella stessa applicazione del dataset client (o del broker XML) oppure può risiedere in un application server come parte di un'applicazione multi-tier.

Questo capitolo descrive come utilizzare un componente provider per controllare l'interazione con i dataset client o con i broker XML.

Individuazione della sorgente dati

Quando si utilizza un componente provider, occorre specificare la sorgente che esso utilizza per ottenere i dati che assemblerà in pacchetti dati. A seconda della versione di C++Builder, è possibile specificare la sorgente come segue:

- Per fornire i dati a partire da un dataset, utilizzare *TDataSetProvider*.
- Per fornire i dati a partire da un documento XML, utilizzare *TXMLTransformProvider*.

Uso di un dataset come sorgente dati

Se il provider è un provider di dataset (*TDataSetProvider*), impostare la proprietà *DataSet* del provider indicando il nome del dataset da utilizzare. In fase di progettazione, selezionare nell'Object Inspector uno dei datasets disponibili nell'elenco a discesa della proprietà *DataSet*.

TDataSetProvider interagisce con il dataset sorgente utilizzando l'interfaccia *IProviderSupport*. Questa interfaccia è resa disponibile da *TDataSet*, ed è perciò disponibile per tutti i dataset. Tuttavia, i metodi di *IProviderSupport* implementati in *TDataSet* sono soprattutto stub il cui unico compito è quello di sollevare eccezioni.

Le classi dataset fornite con C++Builder (i dataset BDE, i dataset ADO, i dataset Client, i dataset *dbExpress* e i dataset Interbase Express) ridefiniscono questi metodi per implementare l'interfaccia *IProviderSupport* in modo più proficuo. I dataset client non aggiungono niente all'implementazione ereditata di *IProviderSupport*, ma possono essere ancora utilizzati come un dataset sorgente purché la proprietà *ResolveToDataSet* del provider sia impostata a **true**.

Gli autori di componenti che creano propri discendenti custom a partire da *TDataSet*, nel caso i dataset debbano fornire i dati a un provider, devono ridefinire tutti i metodi appropriati di *IProviderSupport*. Se il provider si limita a fornire pacchetti dati con operazioni di sola lettura (ovvero, se non applica aggiornamenti), i metodi di *IProviderSupport* implementati in *TDataSet* possono essere sufficienti.

Uso di un documento XML come sorgente dati

Se il provider è un provider XML, impostare la proprietà *XMLDataFile* del provider per indicare il documento di origine.

I provider XML devono trasformare il documento di origine in pacchetti dati, cosicché oltre all'indicazione del documento di origine, si deve anche specificare

come trasformare quel documento in pacchetti dati. Questa trasformazione è gestita dalla proprietà *TransformRead* del provider. *TransformRead* rappresenta un oggetto *TXMLTransform*. È possibile impostarne le proprietà per specificare quale trasformazione utilizzare e utilizzarne gli eventi per fornire il proprio input alla trasformazione. Per ulteriori informazioni sull'utilizzo di provider XML, consultare "Impiego di un documento XML come origine di un provider" a pagina 30-8.

Comunicazione con il dataset clientt

Tutte le comunicazioni fra un provider e un dataset client o un broker XML avvengono attraverso un'interfaccia *IAppServer*. Se il provider è nella stessa applicazione del client, questa interfaccia viene implementata da un oggetto nascosto generato automaticamente, o da un componente *TLocalConnection*. Se il provider fa parte di un'applicazione multi-tier, questa è l'interfaccia del server dell'applicazione oppure (in caso di server SOAP) un'interfaccia generata dal componente connection.

La maggior parte delle applicazioni non utilizzano IAppServerdirettamente, ma lo richiamano indirettamente attraverso le proprietà e i metodi del dataset client o del broker XML. Tuttavia, se occorre, è possibile effettuare chiamate dirette all'interfaccia *AppServer* utilizzando la proprietà AppServerdel dataset client.

La Tabella 28.1 elenca i metodi dell'interfaccia *IAppServer* e i metodi e gli eventi corrispondenti del componente provider e del dataset client. Questi metodi di *IAppServer* includono un parametro *Provider*.Nelle applicazioni multi-tier, questo parametro indica il provider sull'application server con cui il dataset client comunica. Molti metodi includono anche un parametro OleVariant di nome OwnerDatache permette a un dataset client e a un provider di passarsi informazioni custom. Per impostazione predefinita *OwnerData* non è utilizzato, ma viene passato a tutti i gestori di evento rendendo possibile la scrittura di codice che permetta al provider di adattarsi alle informazioni definite dall'applicazione prima e dopo ogni chiamata da parte di un dataset client.

Tabella 28.1 Membri dell'interfaccia AppServer

IAppServer	Componente Provider	TClientDataSet
Metodo AS_ApplyUpdates	Metodo ApplyUpdates, evento BeforeApplyUpdates, evento AfterApplyUpdates	Metodo ApplyUpdates, evento BeforeApplyUpdates, evento AfterApplyUpdates.
Metodo AS_DataRequest	Metodo DataRequest, evento OnDataRequest	Metodo DataRequest.
Metodo AS_Execute	Metodo Execute, evento BeforeExecute, evento AfterExecute	Metodo Execute, evento BeforeExecute, evento AfterExecute.
Metodo AS_GetParams	Metodo GetParams, evento BeforeGetParams, evento AfterGetParams	Metodo FetchParams, evento BeforeGetParams, evento AfterGetParams.
Metodo AS_GetProviderNames	Utilizzato per identificare tutti i provider disponibili.	Utilizzato per creare una lista da usare in progettazione per la proprietà ProviderName.

Tabella 28.1 Membri dell'interfaccia AppServer

IAppServer	Componente Provider	TClientDataSet
Metodo AS_GetRecords	Metodo GetRecords, evento BeforeGetRecords, evento AfterGetRecords	Metodo GetNextPacket, proprietà Data, evento BeforeGetRecords, evento AfterGetRecords
Metodo AS_RowRequest	Metodo RowRequest, evento BeforeRowRequest, evento AfterRowRequest	Metodo FetchBlobs, Metodo FetchDetails, Metodo RefreshRecord, evento BeforeRowRequest, evento AfterRowRequest

Come applicare gli aggiornamenti usando un provider di dataset

I componenti *TXMLTransformProvider* applicano sempre gli aggiornamenti al documento XML associato. Quando si utilizza *TDataSetProvider*, tuttavia, è possibile scegliere come applicare gli aggiornamenti. Per impostazione predefinita, quando i componenti *TDataSetProvider* applicano aggiornamenti e risolvono gli errori di aggiornamento, comunicano direttamente col server di database utilizzando istruzioni SQL generate dinamicamente. Il vantaggio offerto da questo metodo sta nel fatto che l'applicazione server non deve combinare gli aggiornamenti due volte (prima sul dataset e quindi sul server remoto).

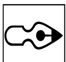
Tuttavia, non è detto che si voglia utilizzare sempre questo approccio. Per esempio, potrebbe essere necessario utilizzare alcuni eventi del componente dataset. Oppure, il dataset utilizzato potrebbe non supportare l'uso di istruzioni SQL (per esempio, se si stanno fornendo dati da un componente *TClientDataSet*).

TDataSetProvider consente di decidete se applicare gli aggiornamenti al server di database, utilizzando istruzioni SQL, o al dataset sorgente, impostando la proprietà *ResolveToDataSet*. Se questa proprietà è impostata a **true**, gli aggiornamenti vengono applicati al dataset. Se è impostata a **false**, gli aggiornamenti vengono applicati direttamente al server di database associato.

Controllo delle informazioni incluse nei pacchetti dati

Operando con un provider di dataset, esistono diversi modi per controllare quali informazioni vengono incluse nei pacchetti dati trasmessi e ricevuti dal client. Questi includono

- Specifica dei campi inclusi in un pacchetto dati
- Impostazione di opzioni che influenzano i pacchetti dati
- Aggiunta di informazioni personali al pacchetto dati

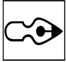


Queste tecniche per il controllo del contenuto di pacchetti dati sono disponibili solo per i provider di dataset. Quando si utilizza *TXMLTransformProvider*, è possibile controllare il contenuto dei pacchetti dati solo controllando il file di trasformazione utilizzato dal provider.

Specifica dei campi inclusi in un pacchetto dati

Operando con un provider di dataset, è possibile controllare quali campi sono inclusi nei pacchetti dati mediante la creazione di campi persistenti nel dataset utilizzato dal provider per generare i pacchetti dati. Il provider includerà quindi solamente questi campi. Possono essere inclusi anche i campi il cui valore viene generato dinamicamente dal dataset sorgente (come i campi calcolati o i campi di consultazione), ma appaiono ai dataset client sul lato ricevente come campi statici a sola lettura. Per informazioni sui campi persistenti, consultare “Componenti campo persistenti” a pagina 23-3.

Se i dataset client dovranno modificare i dati e applicare gli aggiornamenti, sarà necessario includere un numero sufficiente di campi in modo da non avere record duplicati nel pacchetto dati. In caso contrario, quando si applicheranno gli aggiornamenti, sarà impossibile determinare quale record aggiornare. Se si vuole evitare che il dataset client possa vedere o utilizzare quei campi aggiuntivi, forniti al solo scopo di grantire l’unicità dei record, impostare la proprietà *ProviderFlags* di quei campi in modo da includere *pfHidden*.



L’inclusione di un numero di campi sufficiente ad evitare record duplicati è da prendere in considerazione anche quando il dataset sorgente del provider rappresenta una query. Si deve specificare la query in modo da includere abbastanza campi da rendere i record unici, anche se l’applicazione non utilizza tutti i campi.

Impostazione di opzioni che influenzano i pacchetti dati

La proprietà *Options* di un provider di dataset consente di specificare, in fase di trasmissione di BLOB o di tabelle di dettaglio annidate, se includere proprietà di visualizzazione dei campi, che tipi di aggiornamenti sono consentiti e così via. La tabella seguente elenca i possibili valori che possono essere inclusi in *Options*.

Tabella 28.2 Opzione del provider

Valore	Significato
poAutoRefresh	Il provider aggiorna il dataset client con i valori di record correnti non appena ha applicato gli aggiornamenti.
poReadOnly	Il dataset client non può applicare gli aggiornamenti al provider.
poDisableEdits	I dataset client non possono modificare i valori dei dati esistenti. Se l’utente prova a modificare un campo, il dataset client solleva un’eccezione. (Questo non influisce sulla capacità del dataset client di inserire o di cancellare record).
poDisableInserts	I dataset client non possono inserire nuovi record. Se l’utente prova a inserire un nuovo record, il dataset client solleva un’eccezione. (Questo non influisce sulla capacità del dataset client di cancellare i record o di modificare i dati esistenti)
poDisableDeletes	I dataset client non possono cancellare record. Se l’utente prova a cancellare un record, il dataset client solleva un’eccezione. (Questo non influisce sulla capacità del dataset client di inserire o di modificare record)

Tabella 28.2 Opzione del provider

Valore	Significato
poFetchBlobsOnDemand	I valori dei campi BLOB non sono inclusi nel pacchetto dati.Invece, i dataset client devono richiedere questi valori man mano che occorrono. Se la proprietà *FetchOnDemand* del dataset client è **true**, il dataset client richiede questi valori automaticamente. Altrimenti, l'applicazione deve chiamare il metodo *FetchBlobs* del dataset client per reperire i dati BLOB.
poFetchDetailsOnDemand	Quando il dataset del provider rappresenta il master in una relazione master/detail, i valori annidati di dettaglio non sono inclusi nel pacchetto dati. Invece, i dataset client richiedono questi valori man mano che occorrono. Se la proprietà *FetchOnDemand* del dataset client è **true**, il dataset client richiede questi valori automaticamente. Altrimenti, l'applicazione deve chiamare il metodo *FetchDetails* del dataset client per reperire i dettagli annidati.
poIncFieldProps	Il pacchetto dati include, se possibile, le seguenti proprietà di campo: *Alignment*, *DisplayLabel*, *DisplayWidth*, *Visible*, *DisplayFormat*, *EditFormat*, *MaxValue*, *MinValue*, *Currency*, *EditMask*, *DisplayValues.*
poCascadeDeletes	Quando il dataset del provider rappresenta il master in una relazione master/detail, il server cancella automaticamente i record di dettaglio quando vengono cancellati i record principali. Per usare questa opzione, il database server deve essere impostato per eseguire cancellazioni in cascata come parte della sua integrità referenziale.
poCascadeUpdates	Quando il dataset del provider rappresenta il master in una relazione master/detail, i valori chiave delle tabelle di dettaglio vengono automaticamente aggiornati quando vengono modificati i valori corrispondenti nei record del master. Per usare questa opzione, il database server deve essere impostato per eseguire aggiornamenti in cascata come parte della sua integrità referenziale.
poAllowMultiRecordUpdates	Un singolo aggiornamento può provocare la modifica di più record della tabella di database sottostante. Questo comportamento può essere il risultato di trigger, dell'integrità referenziale, di istruzioni SQL sul dataset sorgente, e così via. Si noti che, nel caso si verifichi un errore, i gestori di evento consentono l'accesso al record che è stato aggiornato, ma non agli altri record modificati di conseguenza.
poNoReset	I dataset client non possono specificare che il provider dovrebbe riposizionare il cursore sul primo record prima di reperire i dati.
poPropogateChanges	Le modifiche apportate dal server ai record aggiornati durante il processo di aggiornamento vengono ritrasferite al client e combinate nel dataset del client.
poAllowCommandText	Il client può ridefinire il testo del comando SQL associato del dataset o il nome della tabella o la procedura registrata che esso rappresenta.
poRetainServerOrder	Il dataset client non dovrebbe riordinare i record nel dataset per imporre un ordine predefinito.

Aggiunta di informazioni personali ai pacchetti dati

Usando l'evento *OnGetDataSetProperties*, i provider di dataset possono aggiungere ai pacchetti dati informazioni definite dall'applicazione. Queste informazioni vengono codificate come OleVariant, e memorizzate con un nome specificato. I dataset client possono quindi recuperare le informazioni usando il loro metodo *GetOptionalParam*. È anche possibile specificare di includere le informazioni nei pacchetti delta che il dataset client invia quando vengono aggiornati i record. In questo caso, il dataset client può non essere mai a conoscenza delle informazioni, ma il provider può inviare a se stesso un messaggio con ritorno.

Quando nell'evento *OnGetDataSetProperties* si aggiungono informazioni personali, ogni singolo attributo (a volta chiamato "parametro facoltativo") viene specificato usando un array Variant che contiene tre elementi: il nome (una stringa), il valore (un Variant), e un flag booleano che indica se le informazioni devono essere incluse nei pacchetti delta quando il client applica gli aggiornamenti.Creando un array Variant di array Variant, si possono aggiungere più attributi. Per esempio, il seguente gestore di evento *OnGetDataSetProperties* invia due valori, l'ora in cui i dati sono stati forniti e il numero totale di record del dataset sorgente. Quando i dataset client applicano gli aggiornamenti, vengono restituite solo le informazioni relative all'ora in cui i dati sono stati forniti:

```
void __fastcall TMyDataModule1::Provider1GetDataSetProperties(TObject *Sender, TDataSet
*DataSet, out OleVariant Properties)
{
  int ArrayBounds[2];
  ArrayBounds[0] = 0;
  ArrayBounds[1] = 1;
  Properties = VarArrayCreate(ArrayBounds, 1, varVariant);
  Variant values[3];
  values[0] = Variant("TimeProvided");
  values[1] = Variant(Now());
  values[2] = Variant(true);
  Properties[0] = VarArrayOf(values,2);
  values[0] = Variant("TableSize");
  values[1] = Variant(DataSet->RecordCount);
  values[2] = Variant(false);
  Properties[1] = VarArrayOf(values,2);
}
```

Quando il dataset client applica gli aggiornamenti, l'ora in cui sono stati forniti i record originali può essere letta nell'evento *OnUpdateData* del provider:

```
void __fastcall TMyDataModule1::Provider1UpdateData(TObject *Sender, TCustomClientDataSet
*DataSet)
{
  Variant WhenProvided = DataSet->GetOptionalParam("TimeProvided");
  ...
}
```

Risposta alle richieste di dati del client

Solitamente le richieste di dati da parte del client vengono gestite automaticamente. Un dataset client o un broker XML richiede un pacchetto dati chiamando *GetRecords* (in modo indiretto, mediante l'interfaccia *IAppServer*). Il provider risponde automaticamente prelevando i dati dal dataset o dal documento XML associato, creando un pacchetto dati e inviandolo al client.

Il provider ha la facoltà di modificare i dati dopo che sono stati assemblati in un pacchetto, ma prima che il pacchetto venga inviato al client. Per esempio, si potrebbe voler rimuovere i record dal pacchetto in base a determinati criteri (come il livello di accesso dell'utente), o, in un'applicazione multi-tier, si potrebbe voler crittografare dati particolarmente delicati prima che siano inviato al client.

Per effettuare l'editing del pacchetto dati prima di inviarlo al client, occorre scrivere un gestore di evento *OnGetData* event handler. Il gestore di evento *OnGetData* fornisce il pacchetto dati come un parametro sotto forma di dataset client. Usando i metodi di questo dataset client, è possibile modificare i dati prima di inviarli al client.

Come per tutte le chiamate ai metodi effettuate tramite l'interfaccia *IAppServer*, il provider ha la possibilità di comunicare informazioni di stato persistenti con un dataset client prima e dopo la chiamata a *GetRecords*. Questa comunicazione avviene utilizzando i gestori di evento *BeforeGetRecords* e *AfterGetRecords*. Per una trattazione delle informazioni di stato persistenti negli application server, consultare "Supporto delle informazioni di stato nei moduli dati remoti" a pagina 29-20.

Risposta alle richieste di aggiornamento del client

Un provider applica gli aggiornamenti ai record del database sulla base di un pacchetto dati *Delta* ricevuto da un dataset client o da un broker XML. Il client richiede gli aggiornamenti chiamando il metodo *ApplyUpdates* (in modo indiretto, mediante l'interfaccia *IAppServer*).

Come per tutte le chiamate ai metodi effettuate tramite l'interfaccia *IAppServer*, il provider ha la possibilità di comunicare informazioni di stato persistenti con un dataset client prima e dopo la chiamata a *ApplyUpdates*. Questa comunicazione avviene utilizzando i gestori di evento *BeforeApplyUpdates* e *AfterApplyUpdates*. Per una trattazione delle informazioni di stato persistenti negli application server, consultare "Supporto delle informazioni di stato nei moduli dati remoti" a pagina 29-20.

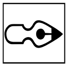
Se si utilizza un provider di dataset, sono disponibili molti altri eventi che consentono un maggore controllo:

Quando un provider di dataset riceve una richiesta di aggiornamento, genera un evento *OnUpdateData*, in cui è possibile modificare il pacchetto Delta prima che venga scritto nel dataset o che influenzi il modo in cui vengono applicati gli aggiornamenti. Dopo l'evento *OnUpdateData*, il provider scrive le modifiche nel database o nel dataset sorgente.

Il componente provider esegue l'aggiornamento record per record. Prima di applicare gli aggiornamenti a ciascun record, il provider genera un evento *BeforeUpdateRecord*, con il quale è possibile vagliare gli aggiornamenti prima che vengano applicati. Se si verifica un errore durante l'aggiornamento di un record, il provider riceve un evento *OnUpdateError* in cui può risolvere l'errore. Solitamente, gli errori si verificano perché la modifica non rispetta un vincolo del server o perché il record del database è stato modificato da un'applicazione differente in una fase successiva al suo recupero da parte del provider, ma prima della richiesta del dataset client di applicare gli aggiornamenti.

Gli errori di aggiornamento possono essere elaborati dal provider di dataset o dal dataset client. Se il provider fa parte di un'applicazione multi-tier dovrebbe gestire tutti gli errori di aggiornamento che non richiedono un'interazione con l'utente per la loro risoluzione. Se il provider non riesce a risolvere una condizione di errore, memorizza temporaneamente una copia del record che ha problemi. Completata l'elaborazione del record il provider restituisce al dataset client il numero di errori incontrati e copia i record non risolti in un pacchetto risultato che viene restituito al dataset client per un'ulteriore riconciliazione.

I set di aggiornamenti vengono trasmessi ai gestori di evento di tutti gli eventi del provider come dataset client. Se il gestore di evento si occupa solo di alcuni tipi di aggiornamenti, è possibile filtrare il dataset in base allo stato di aggiornamento dei record. Filtrando i record, il gestore di evento non deve riordinare i record che non utilizzerà. Per filtrare il dataset client in base allo stato di aggiornamento dei suoi record, impostare la sua proprietà *StatusFilter*.



Le applicazioni devono fornire un supporto più efficace nel caso in cui gli aggiornamenti siano indirizzati su un dataset che non rappresenta una singola tabella. Per i dettagli sulle operazioni da compiere, consultare la sezione "Applicazione di aggiornamenti a dataset che non rappresentano una singola tabella" a pagina 28-12.

Modifica dei pacchetti delta prima dell'aggiornamento del database

Prima di applicare gli aggiornamenti al database, esso genera un evento *OnUpdateData*. Il gestore di evento *OnUpdateData* riceve una copia del pacchetto *Delta* come parametro. Questo è un dataset client.

Nel gestore di evento *OnUpdateData* si può usare qualsiasi proprietà e metodo del dataset client per modificare il pacchetto *Delta* prima che venga scritto nel dataset. Una proprietà particolarmente utile è *UpdateStatus*. *UpdateStatus* indica quale tipo di modifica rappresenta il record corrente del pacchetto delta. Può assumere uno dei valori elencati nella Tabella 28.3.

Tabella 28.3 Valori di UpdateStatus

Valore	Descrizione
usUnmodified	Il contenuto del record non è stato modificato
usModified	Il contenuto del record è stato modificato

Tabella 28.3 Valori di UpdateStatus

Valore	Descrizione
usInserted	Il record è stato inserito
usDeleted	Il record è stato cancellato

Per esempio, il seguente gestore di evento *OnUpdateData* inserisce la data corrente in ogni nuovo record inserito nel database:

```
void __fastcall TMyDataModule1::Provider1UpdateData(TObject *Sender, TCustomClientDataSet
*DataSet)
{
  DataSet->First();
  while (!DataSet->Eof)
  {
    if (DataSet->UpdateStatus == usInserted)
    {
      DataSet->Edit();
      DataSet->FieldByName("DateCreated")->AsDateTime = Date();
      DataSet->Post();
    }
    DataSet->Next();
  }
}
```

Come influenzare la modalità di applicazione degli aggiornamenti

L'evento *OnUpdateData* dà al provider di dataset anche la possibilità di indicare come i record del pacchetto delta vengono applicati al database.

Per impostazione predefinita, le modifiche nel pacchetto delta vengono scritte nel database usando istruzioni SQL di tipo UPDATE, INSERT o DELETE generate automaticamente; ad esempio

```
UPDATE EMPLOYEES
  set EMPNO = 748, NAME = 'Smith', TITLE = 'Programmer 1', DEPT = 52
WHERE
  EMPNO = 748 and NAME = 'Smith' and TITLE = 'Programmer 1' and DEPT = 47
```

Se non specificato diversamente, tutti i campi dei record del pacchetto delta vengono inclusi nella clausola UPDATE e nella clausola WHERE. Tuttavia, può essere opportuno escludere alcuni di questi campi. Un modo per farlo consiste nell'impostare la proprietà *UpdateMode* del provider. Alla proprietà *UpdateMode* può essere assegnato uno dei seguenti valori:

Tabella 28.4 Valori di UpdateMode

Valore	Significato
upWhereAll	Tutti i campi vengono usati per individuare i campi (clausola WHERE).
upWhereChanged	Per individuare i record vengono usati solo i campi chiave e i campi che sono stati modificati.
upWhereKeyOnly	Per individuare i record vengono usati solo i campi chiave.

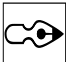
Potrebbe, comunque, essere necessario anche un maggiore controllo. Per esempio, con l'istruzione precedente si potrebbe voler impedire la modifica del campo EMPNO escludendolo dalla clausola UPDATE ed escludere dalla clausola WHERE i campi TITLE e DEPT per evitare conflitti di aggiornamento nel caso altre applicazioni abbiano modificato i dati. Per specificare le clausole in cui appare un certo campo, si deve usare la proprietà *ProviderFlags*. *ProviderFlags* è un set che può includere uno dei valori elencati nella Tabella 28.5

Tabella 28.5 Valori di ProviderFlags

Valore	Descrizione
pfInWhere	Il campo appare nella clausola WHERE delle istruzioni INSERT, DELETE e UPDATE generate quando *UpdateMode* è *upWhereAll* oppure *upWhereChanged*.
pfInUpdate	Il campo appare nella clausola UPDATE delle istruzioni UPDATE generate.
pfInKey	Il campo viene usato nella clausola WHERE delle istruzioni generate quando *UpdateMode* è *upWhereKeyOnly*.
pfHidden	Il campo viene incluso nei record per assicurare l'univocità, ma non può essere visto o usato sul lato client.

Così, il seguente gestore di evento *OnUpdateData* permette di aggiornare il campo TITLE e utilizza i campi EMPNO e DEPT per individuare il record chiesto. Se si verifica un errore, e viene fatto un secondo tentativo per individuare il record solamente in base alla chiave, il comando SQL generato cercherà solo il campo EMPNO:

```
void __fastcall TMyDataModule1::Provider1UpdateData(TObject *Sender, TCustomClientDataSet
*DataSet)
{
  DataSet->FieldByName("EMPNO")->ProviderFlags.Clear();
  DataSet->FieldByName("EMPNO")->ProviderFlags << pfInWHere << pfInKey;
  DataSet->FieldByName("TITLE")->ProviderFlags.Clear();
  DataSet->FieldByName("TITLE")->ProviderFlags << pfInUpdate;
  DataSet->FieldByName("DEPT")->ProviderFlags.Clear();
  DataSet->FieldByName("DEPT")->ProviderFlags << pfInWhere;
}
```



È possibile usare la proprietà *UpdateFlags* per influenzare il modo in cui vengono applicati gli aggiornamenti anche si sta aggiornando un dataset senza usare istruzioni SQL generate dinamicamente. Anche questi flag determinano quali campi vengono usati per individuare i record e quali campi vengono aggiornati.

Vaglio dei singoli aggiornamenti

Immediatamente prima che ciascun aggiornamento venga applicato, il provider di dataset riceve un evento *BeforeUpdateRecord*. Questo evento può essere usato per modificare i record prima vengano applicati, analogamente a come si usa l'evento *OnUpdateData* per modificare interi pacchetti delta. Ad esempio, il provider non confronta i campi BLOB (come i memo) quando esegue il controllo su conflitti di aggiornamento. Se si vogliono controllare eventuali errori di aggiornamento relativi a campi BLOB, è possibile utilizzare l'evento *BeforeUpdateRecord*.

Inoltre, si può usare questo evento per applicare personalmente gli aggiornamenti o per vagliare e rigettare gli aggiornamenti. Il gestore dell'evento *BeforeUpdateRecord* permette di segnalare che un aggiornamento è già stato gestito e non deve essere applicato. Il provider quindi salta quel record, ma non lo considera come un errore di aggiornamento. Per esempio, questo evento fornisce a una procedura memorizzata (che non può essere aggiornata automaticamente) un meccanismo per applicare gli aggiornamenti, consentendo al provider di saltare qualsiasi elaborazione automatica dal momento che il record viene aggiornato dall'interno del gestore di evento.

Risoluzione degli errori di aggiornamento sul provider

Quando si determina una condizione di errore non appena il provider di dataset tenta di inviare un record nel pacchetto delta, si verifica un evento *OnUpdateError*. Se il provider non riesce a risolvere un errore di aggiornamento, memorizza temporaneamente una copia del record che ha generato l'errore. Completata l'elaborazione del record, il provider restituisce il numero di errori incontrati e copia i record non risolti in un pacchetto dati risultato che viene ripassato al client per un'ulteriore riconciliazione.

Nelle applicazioni multi-tiered, questo meccanismo permette di gestire qualsiasi errore di aggiornamento che è possibile risolvere automaticamente sull'application server, pur consentendo ancora l'interazione con l'utente sull'applicazione client per correggere le condizioni di errore.

Il gestore *OnUpdateError* riceve una copia del record che non può essere modificato, un codice di errore dal database e un'indicazione del fatto che il resolver stesse cercando di inserire, cancellare o aggiornare il record. Il record problematico viene restituito in un dataset client. Su questo dataset non si devono mai usare i metodi per lo spostamento tra i dati. Tuttavia, per ciascun campo del dataset, è possibile usare le proprietà *NewValue*, *OldValue* e *CurValue* per determinare la causa del problema e apportare tutte le modifiche necessarie per risolvere l'errore di aggiornamento. Se il gestore di evento *OnUpdateError* può correggere il problema, imposta il parametro *Response* in modo che venga applicato il record corretto.

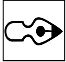
Applicazione di aggiornamenti a dataset che non rappresentano una singola tabella

Quando un provider di dataset genera istruzioni SQL che applicano gli aggiornamenti direttamente su un server di database, è necessario fornire il nome della tabella di database che contiene i record. Questo compito può essere gestito in modo automatico per molti dataset, come i dataset di tipo tabella o i componenti *TQuery* aggiornati in diretta. Gli aggiornamenti automatici, tuttavia, rappresentano un problema nel caso il provider debba applicare gli aggiornamenti ai dati sottostanti una procedura registrata a partire da un set risultato o da una query su più tabelle. Non esiste un modo semplice per accertare il nome della tabella a cui dovrebbero essere applicati gli aggiornamenti.

Se la query o la procedura registrata è un dataset BDE (*TQuery* o *TStoredProc*) a cui è associato un oggetto update, il provider utilizzerà l'oggetto update. Tuttavia, se non

esiste un oggetto update, è possibile fornire da programma il nome della tabella in un gestore di evento *OnGetTableName*. Una volta che il gestore di evento ha fornito il nome della tabella, il provider è in grado di generare le opportune istruzioni SQL per applicare gli aggiornamenti.

Il passaggio del nome della tabella funziona solamente se il destinatario degli aggiornamenti è una singola tabella di database (vale a dire, è necessario aggiornare solo i record di una tabella). Se gli aggiornamenti devono apportare modifiche a più tabelle di database sottostanti, è necessario applicare gli aggiornamenti in modo esplicito nel codice utilizzando l'evento *BeforeUpdateRecord* del provider. Dopo che il gestore di evento ha applicato un aggiornamento, si deve impostare a **true** il parametro *Applied* del gestore di evento in modo che il provider non generi un errore.



Se il provider è associato a un dataset BDE, è possibile utilizzare nel gestore di evento *BeforeUpdateRecord* un oggetto update per applicare gli aggiornamenti mediante istruzioni SQL personalizzate. Per i dettagli, consultare "Aggiornamento di dataset mediante oggetti update" a pagina 24-41 for details.

Risposta agli eventi generati dal client

I componenti provider implementano un evento di tipo generico che permette di creare proprie chiamate dai dataset client direttamente al provider. Si tratta dell'evento *OnDataRequest*.

OnDataRequest non fa parte del normale funzionamento del provider. È solo un aggancio per consentire ai dataset client di comunicare direttamente con i provider. Il gestore di evento accetta come parametro in ingresso un OleVariant e restituisce un OleVariant. Usando gli OleVariant, l'interfaccia è sufficientemente generica per contenere quasi tutte le informazioni che si vogliono passare al o dal provider.

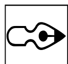
Per generare un evento *OnDataRequest*, l'applicazione client chiama direttamente il metodo *DataRequest* del dataset client.

Gestione dei vincoli del server

La maggior parte dei sistemi di gestione dei database relazionali implementa vincoli sulle tabelle per far rispettare l'integrità dei dati. Un vincolo è una regola che governa i valori dei dati nelle tabelle e nelle colonne, o le relazioni tra i dati tra le colonne di tabelle differenti. Per esempio, la maggior parte dei database relazionali conformi a SQL-92 supporta i seguenti vincoli:

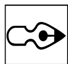
- NOT NULL, per garantire che un valore fornito a una colonna abbia un valore.
- NOT NULL UNIQUE, per garantire che il valore della colonna abbia un valore e non duplichi un altro valore presente in quella colonna per un altro record.
- CHECK, per garantire che un valore fornito a una colonna cada all'interno di un determinato intervallo, o appartenga a un insieme finito di possibili valori.

- CONSTRAINT, un vincolo di controllo a livello di tabella che si applica a più colonne.
- PRIMARY KEY, per designare una o più colonne come chiave principale della tabella per l'indicizzazione.
- FOREIGN KEY, per designare una o più colonne di una tabella che fanno riferimento a un'altra tabella.



Questo elenco non è esclusivo. Il server di database può supportare alcuni o tutti questi vincoli, in parte o completamente, nonché ulteriori vincoli. Per ulteriori informazioni sui vincoli supportati, consultare la documentazione del server.

I vincoli del database server duplicano ovviamente molti tipi di controlli sui dati normalmente gestiti dalle tradizionali applicazioni database desktop. Nelle applicazioni database multi-tier è possibile sfruttare i vantaggi dei vincoli del server Se il provider opera con un dataset BDE, la proprietà *Constraints* consente di replicare e applicare i vincoli del server ai dati passati e ricevuti dai dataset client. Se *Constraints* è **true** (impostazione predefinita), i vincoli del server memorizzati nel dataset sorgente vengono inclusi nei pacchetti dati e hanno effetto sui tentativi del client di aggiornare i dati.



Prima che il provider server possa trasferire ai dataset client le informazioni sui vincoli, deve recuperare i vincoli dal server di database. Per ottenere dal server i vincoli sul database, utilizzare SQL Explorer per importare nel Data Dictionary i vincoli del server di database e le espressioni predefinite. I vincoli e le espressioni predefinite inclusi nel Data Dictionary sono resi automaticamente disponibili ai dataset BDE.

A volte si potrebbe non voler applicare i vincoli del server ai dati inviati a un dataset client. Per esempio, un dataset client che riceve dati in pacchetti e consente l'aggiornamento in locale dei record prima di prelevare altri record, potrebbe aver bisogno di disattivare alcuni vincoli del server che potrebbero essere attivati per la presenza di un set di dati momentaneamente incompleto. Per impedire la replica dei vincoli dal provider al dataset client, impostare *Constraints* a **false**. Si noti, inoltre, che i dataset client possono disattivare e attivare i vincoli usando i metodi *DisableConstraints* e *EnableConstraints*. Per ulteriori informazioni sull'attivazione e sulla disattivazione dei vincoli da parte del dataset client, consultare "Gestione dei vincoli dal server" a pagina 27-32.

Capitolo

29

Creazione di applicazioni multi-tier

Questo capitolo descrive come creare un'applicazione database multi-tier client/server. Un'applicazione multi-tier client/server è partizionata in unità logiche che vengono eseguite insieme su macchine distinte. Le applicazioni multi-tier condividono i dati e comunicano con le altre applicazioni in una rete locale o anche su Internet. Esse forniscono molti vantaggi, come la logica di gestione centralizzata e le applicazioni thin client.

Nella sua forma più semplice, detta anche "modello three-tier", un'applicazione multi-tier è suddivisa in tre parti:

- **Applicazione client**: fornisce un'interfaccia utente nella macchina dell'utente stesso.
- **Application server**: risiede in una postazione centrale della rete accessibile a tutti i client e fornisce servizi comuni per i dati.
- **Server di database remoto**: fornisce il sistema relazionale di gestione del database (RDBMS)

In questo modello three-tier l'application server gestisce il flusso dei dati tra i client e il database server remoto, perciò a volte viene chiamato "data broker". Solitamente si creano solo l'application server e i suoi client, ma, volendo, si può anche creare un database di back end personalizzato.

Nelle applicazioni multi-tier più complesse i servizi aggiuntivi risiedono tra un client e un database server remoto. Per esempio, potrebbero esservi un broker di servizi di sicurezza per gestire in modo sicuro le transazioni Internet, o dei servizi di bridge per gestire la condivisione dei dati con database su altre piattaforme.

Il supporto della VCL e della CLX per lo sviluppo di applicazioni multi-tier è un ampliamento del modo in cui i dataset client comunicano con un componente provider utilizzando pacchetti dati trasportabili. In questo capitolo viene specificatamente descritta la creazione di un'applicazione database three-tier. Dopo aver capito come creare e gestire un'applicazione three-tier, si può passare a creare e ad aggiungere ulteriori livelli di servizi, a seconda delle necessità.

Vantaggi di un modello database multi-tier

Il modello database multi-tier suddivide un'applicazione database in parti logiche. L'applicazione client può occuparsi della visualizzazione dei dati e delle interazioni con l'utente. Teoricamente, l'applicazione non sa nulla del modo in cui vengono memorizzati o conservati i dati. L'application server (il tier intermedio) coordina ed elabora le richieste e gli aggiornamenti da parte di più client. Gestisce, inoltre, tutti i dettagli della definizione dei dataset e dell'interazione con il server di database.

I vantaggi di questo modello multi-tier comprendono:

- **Incapsulamento della logica di gestione in un tier intermedio condiviso**. Applicazioni client differenti accedono tutte alle stesso tier intermedio. Ciò consente di evitare la ridondanza (e i costi di manutenzione) della duplicazione delle regole di gestione per ogni singola applicazione client.
- **Applicazioni thin client**. Le applicazioni client possono essere scritte in modo da occupare poco spazio, delegando la maggior parte dell'elaborazione ai tier intermedi. Non solo si hanno applicazioni client più piccole, ma anche più facili da distribuire, in quanto non è necessario preoccuparsi della loro installazione, configurazione e della manutenzione del software per la connettività al database (come con Borland Database Engine e con i server di database lato client). Le applicazioni thin client possono anche essere distribuite su Internet per una maggior flessibilità.
- **Elaborazione distribuita dei dati**. Distribuendo il lavoro di un'applicazione su più macchine, è possibile migliorare le prestazioni grazie al bilanciamento del carico, e consentire ai sistemi ridondanti di subentrare quando un server si blocca.
- **Maggiori possibilità di sicurezza**. È possibile isolare le funzioni riservate e delicate in tier che hanno restrizioni di accesso differenti. Ciò fornisce livelli flessibili e configurabili di sicurezza. I tier intermedi possono limitare i punti di accesso al materiale riservato, consentendo di controllare più facilmente l'accesso. Se si usa HTTP o COM+, si possono sfruttare i vantaggi dei modelli di sicurezza che tali tecnologie supportano.

Applicazioni multi-tier basate su provider

Le applicazioni multi-tier si basano sull'uso dei componenti delle pagine DataSnap, Data Access e possibilmente WebServices della Component palette, oltre a un modulo dati remoto che viene creato da un wizard presente sulla pagina Multitier o WebServices della finestra di dialogo New Items. Essi si basano sulla capacità dei componenti provider di assemblare dati in pacchetti dati trasportabili e di gestire gli aggiornamenti ricevuti come pacchetti delta trasportabili.

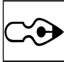
I componenti necessari per un'applicazione multi-tier sono descritti nella Tabella 29.1:

Tabella 29.1 Componenti usati nelle applicazioni multi-tier

Componente	Descrizione
Moduli dati remoti	Moduli dati specialistici che operano con Automation server COM e con un'applicazione Web Service per fornire l'accesso delle applicazioni client a tutti i provider che contengono. Vengono usati negli application server.
Componente Provider	Un broker di dati che fornisce i dati tramite la creazione di pacchetti dati e risolve gli aggiornamenti del client. Viene usato negli application server.
Componente dataset Client	Un dataset specializzato che usa midas.dll per gestire i dati memorizzati nei pacchetti dati. Il dataset client è utilizzato nell'applicazione client. Memorizza gli aggiornamenti nella cache locale e li applica in pacchetti delta all'application server.
Componenti connection	Una famiglia di componenti che individua il server, le connessioni della scheda e rende disponibile ai dataset client l'interfaccia *IAppServer*. Ogni componente connection è specializzato per usare un particolare protocollo di comunicazione.

Il provider e i componenti dataset client necessitano di midas.dll o midaslib.dcu, che gestisce i dataset memorizzati come pacchetti dati. (Notare che, poiché il provider è utilizzato sull'application server e il dataset client è utilizzato sull'applicazione client, se si utilizza midas.dll, è necessario distribuirla sia sull'application server che sull'applicazione client.)

Se si utilizzano dataset BDE, l'application server potrebbe richiedere anche l'uso di SQL Explorer per agevolare l'amministrazione del database e per importare nel Data Dictionary i vincoli del server, in modo da poterli controllare a qualsiasi livello dell'applicazione multi-tier.



Per distribuire l'application server è necessario acquistare le licenze per il server.

Una panoramica dell'architettura in cui sono inseriti questi componenti è presentata in "Utilizzo di un'architettura multi-tier" a pagina 18-13.

Panoramica di un'applicazione three-tier

Le seguenti fasi illustrano una normale sequenza di eventi per un'applicazione three-tier basata su provider:

1. Un utente avvia l'applicazione client. Il client si collega all'application server (che può essere specificato in fase di progettazione o di esecuzione). Se l'application server non è già in esecuzione, viene avviato. Il client riceve un'interfaccia *IAppServer* per la comunicazione con l'application server.
2. Il client richiede dati dall'application server. Un client può richiedere i dati in un'unica soluzione o pochi per la volta per tutta la sessione (prelievo su richiesta).
3. L'application server recupera i dati (stabilendo innanzi tutto, se necessario, una connessione al database), crea i pacchetti dati per il client e li restituisce al client. Altre informazioni, (ad esempio, le caratteristiche di visalizzazione dei campi)

possono essere incluse nei metadati del pacchetto dati. Questo processo di inglobamento dei dati in pacchetti viene chiamato “fornitura”.

4 Il client decodifica il pacchetto dati e visualizza i dati all’utente.

5 Man mano che l’utente interagisce con l’applicazione client, i dati vengono aggiornati (vengono aggiunti, cancellati o modificati record). Queste modifiche vengono memorizzate in un registro delle modifiche dal client.

6 Infine il client applica i suoi aggiornamenti all’application server, normalmente in risposta a un’azione dell’utente. Per applicare questi aggiornamenti, il client compone in un pacchetto il suo registro delle modifiche e lo invia come un pacchetto dati al server.

7 L’application server decodifica il pacchetto dati e invia gli aggiornamenti (se è il caso, nel contesto di una transazione). Se un record non può essere inviato al server (per esempio, perché un’altra applicazione ha modificato il record, dopo che il record era stato richiesto da un client e prima che quest’ultimo applicasse i suoi aggiornamenti), l’application server tenta di riconciliare le modifiche del client con i dati correnti, oppure salva i record che potrebbero non essere inoltrati. Questo processo di invio dei record e di memorizzazione nella cache dei record problematici viene chiamato “risoluzione”.

8 Quando l’application server termina il processo di risoluzione, restituisce tutti i record non inoltrati al client per un’ulteriore risoluzione.

9 Il client riconcilia i record non risolti. Esistono molti modi in cui un client può riconciliare i record non risolti. Normalmente, il client cerca di correggere la situazione che impedisce ai record di essere inviati o scarta le modifiche. Se la situazione di errore può essere rettificata, il client applica di nuovo gli aggiornamenti.

10 Il client effettua l’aggiornamento dei suoi dati dal server.

Struttura dell’applicazione client

Per l’utente finale l’applicazione client di un’applicazione multi-tier appare e si comporta in modo non diverso da una tradizionale applicazione two-tier che utilizza aggiornamenti in cache. L’interazione con l’utente ha luogo tramite i controlli standard associati ai dati, che visualizzano i dati da un componente *TClientDataSet*. Per informazioni dettagliate sull’uso delle proprietà, degli eventi e dei metodi dei dataset client, consultare il Capitolo 27, “Utilizzo dei dataset client”.

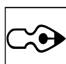
TClientDataSet preleva i dati da un componente provider e vi applica gli aggiornamenti, esattamente come nelle applicazioni two-tier che utilizzano un dataset client con un provider esterno. Per i dettagli sui componenti provider, vedere il Capitolo 28, “Uso di componenti provider”. Per i dettagli sulle funzioni dei dataset client che ne agevolano la comunicazione con un provider, vedere “Uso di un dataset client con un provider di dati” a pagina 27-26

Il dataset client comunica con il provider attraverso l’interfaccia *IAppServer*. Esso ottiene questa interfaccia da un componente connection. Il componente connection stabilisce la connessione all’application server. Sono disponibili differenti

componenti connection per usare protocolli di comunicazione diversi. Questi componenti connection sono riassunti nella seguente tabella:

Tabella 29.2 Componenti connection

Componente	Protocollo
TDCOMConnection	DCOM
TSocketConnection	Windows sockets (TCP/IP)
TWebConnection	HTTP
TSOAPConnection	SOAP (HTTP and XML)



La pagina DataSnap della Component palette include anche un componente connection che non si collega affatto a un application server, ma fornisce invece un'interfaccia *IAppServer* che deve essere utilizzata dai dataset client durante la comunicazione con provider presenti nella stessa applicazione. Questo componente, *TLocalConnection*, non è necessario, ma semplifica un successivo ampliamento a multi-tier dell'applicazione.

Per ulteriori informazioni sull'uso dei componenti connection, consultare "Connessione all'application server" a pagina 29-24.

La struttura dell'application server

Quando un application server viene impostato ed eseguito, non stabilisce alcuna connessione con le applicazioni client. Sono invece le applicazioni client che iniziano e mantengono la connessione. Un'applicazione client usa un componente connection per stabilire una connessione con l'application server e utilizza l'interfaccia dell'application server per comunicare con un provider selezionato. Tutto ciò avviene automaticamente, senza che si debba scrivere del codice per gestire le richieste in arrivo o fornire le interfacce.

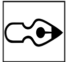
La base di un application server è un modulo dati remoto, che è un modulo dati specializzato che supporta l'interfaccia *IAppServer* (per gli application server che funzionano anche come Web Service, il modulo dati remoto supporta anche l'interfaccia *IAppServerSOAP* e la preferisce a *IAppServer*.) Le applicazioni client utilizzano l'interfaccia del modulo dati remoto per comunicare con i provider sull'application server. Quando il modulo dati remoto usa *IAppServerSOAP*, il componente connection lo adatta a un'interfaccia di *IAppServer* che i dataset client sono in grado di utilizzare.

Esistono due tipi di moduli dati remoto:

- I moduli dati remoti basati su COM sono moduli dati che utilizzano un oggetto associato, detto oggetto di implementazione, che deriva da REMOTEDATAMODULE_IMPL(). REMOTEDATAMODULE_IMPL è una macro definita in Atlvcl.h che elenca gli antenati dell'oggetto di implementazione. Questi includono le classi *CComObjectRootEx* e *CComCoClass* di ATL, oltre che l'interfaccia *IAppServer*. Se si sta creando un application server che sia in grado di sfruttare i servizi distribuiti per le applicazioni forniti da MTS o COM+, REMOTEDATAMODULE_IMPL include inoltre l'interfaccia *IObjectControl*,

necessaria per tutti gli oggetti transazionali. La classe di implementazione, generata automaticamente da un wizard, ha accesso a un'interfaccia *IObjectContext*, messa a disposizione dal sistema per suo conto, e che viene utilizzata dall'interfaccia per gestire transazioni, liberare risosrse o sfruttare il supporto per la sicurezza.

- I moduli dati SOAP sono moduli dati che implementano l'interfaccia *IAppServerSOAP* come interfaccia richiamabile. Questi moduli dati vengono aggiunti a un'applicazione Web Service e consentono ai client di accedere ai dati come un Web Service. Per informazioni sulle applicazioni Web Service, vedere il Capitolo 36, "Utilizzo dei Web Services".



Se l'application server deve essere distribuito in ambiente MTS o COM+, il modulo dati remoto include degli eventi legati all'attivazione o alla disattivazione dell'application server. Ciò consente all'application server di acquisire le connessioni ai database al momento della sua attivazione e di rilasciarle quando viene disattivato.

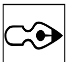
Contenuti del modulo dati remoto

Come con qualsiasi altro modulo dati, nel modulo dati remoto è possibile includere qualsiasi componente non visuale. Vi sono tuttavia alcuni componenti che è necessario includere:

- Se il modulo dati remoto espone le informazioni provenienti da un server di database, deve includere un componente dataset per rappresentare i record da quel server di database. Possono essere necessari altri componenti, come un componente database connection di qualsiasi tipo, per permettere al dataset di interagire con un server di database. Per informazioni sui dataset, consultare il Capitolo 22, "I dataset". Per informazioni sui componenti database connection, veder il Capitolo 21, "Connessione ai database".

  Per ogni dataset che il modulo dati remoto espone ai client, deve includere un provider di dataset. Un provider di dataset assembla i dati in pacchetti dati che vengono inviati ai dataset client e applica aggiornamenti ricevuti dai dataset client a un dataset sorgente o a un server di database. Per ulteriori informazioni sui provider di dataset, vedere il Capitolo 28, "Uso di componenti provider".

- Per ogni documento XML che il modulo dati remoto espone ai client, deve includere un provider XML. Un provider XML agisce come un provider di dataset, salvo che preleva i dati da un documento XML invece che da un server di database e vi applica aggiornamenti. Per ulteriori informazioni sull'utilizzo di provider XML, consultare "Impiego di un documento XML come origine di un provider" a pagina 30-8.



Non confondere un componente per la connessione ai database, che connette i dataset a un server di database, con il componente connection usato dalle applicazioni client nelle applicazioni multi-tier. I componenti connection per le applicazioni multi-tier sono inclusi sulle pagine DataSnap o WebServices della Component palette.

Uso di moduli dati transazionali

È possibile scrivere application server in grado di sfruttare i servizi speciali per le applicazioni distribuite messi a disposizione da MTS (prima di Windows 2000) o da COM+ (in ambiente Windows 2000 e successivi). Per fare ciò, invece di un comune modulo dati remoto, si crea un modulo dati transazionale.

Quando si usa un modulo dati a transazionale, l'applicazione può sfruttare i seguenti servizi speciali:

- **Sicurezza**. MTS e COM+ forniscono all'application server un servizio per la sicurezza basato sui ruoli. Ai client vengono assegnati dei ruoli che determinano se essi possono accedere o meno all'interfaccia dell'application server. Per accedere a questi servizi di sicurezza, è possibile utilizzare l'interfaccia *IObjectContext* a cui si accede mediante la classe di implementazione. Per ulteriori informazioni sulla sicurezza MTS e COM+, consultare il paragrafo "Sicurezza basata sul ruolo" a pagina 44-17.
- **Condivisione degli handle del database**. I moduli dati transazionali condividono automaticamente le connessioni al database effettuate tramite ADO o (se si sta utilizzando MTS e si attiva MTS POOLING) BDE. Quando un client ha completato una connessione al database, un altro client può riutilizzarla. Questa tecnica riduce il traffico di rete, in quanto il tier intermedio non ha bisogno di sconnettersi dal database server remoto e di registrarsi poi di nuovo. Quando si condividono gli handle del database, il componente di conessione al database deve impostare la sua proprietà *KeepConnection* a **false**, in modo che l'applicazione consenta la massima condivisione delle connessioni. Per ulteriori informazioni sulla centralizzazione per la gestione del database, consultare "Distributori di risorse di database" a pagina 44-6.
- **Transazioni**. Quando si utilizza un modulo dati transazionale, è possibile fornire un supporto per le transazioni ancora più evoluto di quello disponibile con una singola connessione database.I moduli dati transazionali possono partecipare a transazioni che coinvolgono più database, o includere funzioni che non coinvolgono affatto i database. Per ulteriori informazioni sul supporto transazionale fornito dagli oggetti transazionali, come i moduli dati transazionali, consultare "Gestione delle transazioni nelle applicazioni multi-tiered" a pagina 29-18.
- **Attivazione just-in-time e disattivazione as-soon-as-possible.** È possibile scrivere il proprio server in modo che le istanze vengano attivate e disattivate quando è necessario.Quando si usa l'attivazione just-in-time e la disattivazione as-soon-as-possible, l'application server viene istanziato solo quando è necessario gestire le richieste del client. Ciò impedisce di bloccare le risorse come gli handle del database quando non vengono usati.

  L'uso dell'attivazione just-in-time e della disattivazione as-soon-as-possible fornisce una soluzione intermedia tra il routing di tutti i client tramite una singola istanza del modulo dati remoto e la creazione di un'istanza distinta per ogni connessione del client. Con un'unica istanza l'application server deve gestire tutte le chiamate al database tramite una singola connessione al database. Ciò agisce come un collo di bottiglia, e può avere effetti sulle prestazioni quando ci sono molti client. Con più istanze del modulo dati remoto, ciascuna istanza può

mantenere una distinta connessione al database, evitando pertanto la necessità di serializzare l'accesso al database. Tuttavia, ciò monopolizza le risorse perché gli altri client non possono usare la connessione al database mentre è associata al modulo dati remoto di un altro client.

Per sfruttare i vantaggi delle transazioni, dell'attivazione just-in-time e della disattivazione as-soon-as-possible, le istanze del modulo dati remoto devono essere senza stato.Ciò significa che occorre fornire un supporto ulteriore nel caso in cui il client si appoggi alle informazioni di stato. Per esempio, durante l'esecuzione di un recupero dati incrementale, il client deve passare le informazioni relative al record corrente. Ciò significa che occorre fornire un supporto ulteriore nel caso in cui il client si appoggi alle informazioni di stato. Per maggiori informazioni sulle informazioni di stato e sui moduli dati remoto nelle applicazioni multi-tier, consultare la sezione "Supporto delle informazioni di stato nei moduli dati remoti" a pagina 29-20.

Per impostazione predefinita, tutte le chiamate generate automaticamente a un modulo dati sono transazionali (cioè , partono dal presupposto che, al termine della chiamata, sia possibile disattivare il modulo dati e confermare o annullare tutte le transazioni attive). È possibile scrivere un modulo dati transazionale, che dipende dalle informazioni di stato persistenti, impostando a **false** la proprietà *AutoComplete* per l'attivazione just-in-time, anche se non supporterà le transazioni, o per la disattivazione as-soon-as-possible a meno che non si usi un'interfaccia custom.



Gli application server contenenti moduli dati transazionali non dovrebbero aprire le connessioni ai database finché non viene attivato il modulo dati. In fase di progettazione bisogna accertarsi che tutti i dataset non siano attivi e che il database non sia collegato prima di eseguire l'applicazione. Nell'applicazione stessa aggiungere il codice, per aprire le connessioni al database quando il modulo dati viene attivato, e chiuderlo quando viene disattivato.

Centralizzazione di moduli dati remoti

La centralizzazione di oggetti consente di creare una cache di application server che vengono condivisi dai rispettivi client, risparmiando in tal modo risorse. La modalità di funzionamento dipende dal tipo di modulo dati remoto e dal protocollo di connessione.

Se si sta creando un modulo dati transazionale che verrà installato in ambiente COM+, è possibile usare il COM+ Component Manager per installare l'application server come oggetto centralizzato. Per maggiori informazioni, consultare "Condivisione degli oggetti" a pagina 44-10.

Anche se non si sta utilizzando un modulo dati transazionale, si può trarre vantaggio dalla centralizzazione degli oggetti se la connessione viene stabilita mediante il protocollo *TWebConnection*. In questo secondo tipo di centralizzazione di oggetti, si limita il numero di istanze che vengono create dell'application server. In questo modo si limita il numero di connessioni database che è necessario conservare, oltre a tutte le altre risorse utilizzate dall'application server.

Quando l'applicazione Web Server (che passa le chiamate all'application server) riceve richieste dal client, le passa al primo application server disponibile nel sistema centralizzato. Se non c'è un application server, disponibile, ne crea uno nuovo (fino al

numero massimo specificato). In questo modo si fornisce una via di mezzo tra l'indirizzamento di tutti i client su una singola istanza dell'application server (che può comportarsi da collo di bottiglia), e la creazione di un'istanza separata per ogni connessione client (che può richiedere molte risorse).

Se un'istanza dell'application server nel sistema centralizzato non riceve nessuna richiesta dai client per un certo periodo di tempo, viene automaticamente resa disponibile. In questo si impedisce al sistema centralizzato di monopolizzare risorse a meno che non siano utilizzate.

Per impostare la centralizzazione degli oggetti utilizzando una connessione Web (HTTP), fare quanto segue:

1 Individuare il metodo *UpdateRegistry* della classe di implementazione. Questo metodo appare nel file header della unit di implementazione:

```
static HRESULT WINAPI UpdateRegistry(BOOL bRegister)
{
  TRemoteDataModuleRegistrar regObj(GetObjectCLSID(), GetProgID(), GetDescription());
  return regObj.UpdateRegistry(bRegister);
}
```

2 Impostare a **true** il flag *RegisterPooled* della variabile *regObj*, che è un'istanza di *TRemoteDataModuleRegistrar*. Potrebbe essere necessario impostare inoltre altre proprietà di *regObj* per indicare in che modo dovrà essere gestita la cache dei moduli dati remoti. Ad esempio, il codice seguente consente un massimo di 10 istanze del modulo dati remoto e le elimina dalla cache nel caso in cui rimangano inutilizzate per più di 15 minuti:

```
static HRESULT WINAPI UpdateRegistry(BOOL bRegister)
{
  TRemoteDataModuleRegistrar regObj(GetObjectCLSID(), GetProgID(), GetDescription());
  regObj.RegisterPooled = true;
  regObj.Timeout = 15;
  regObj.Max = 10;
  return regObj.UpdateRegistry(bRegister);
}
```

Quando si utilizza l'uno o l'altro di questi metodi per la centralizzazione degli oggetti, è necessario che l'application server sia senza stato. Ciò perché una singola istanza è in grado di gestire potenzialmente richieste provenienti da parecchi clients. Se facesse affidamendo alla persistenza delle informazioni di stato, i client potrebbero interferire fra di loro. Per maggiori informazioni su come garantire che il modulo dati remoti sia senza stato, consultare la sezione "Supporto delle informazioni di stato nei moduli dati remoti" a pagina 29-20.

Scelta di un protocollo di connessione

Ogni protocollo di comunicazione che può essere usato per collegare le applicazioni client all'application server fornisce specifici vantaggi. Prima di scegliere un protocollo, si deve considerare il numero di client attesi, come verrà distribuita l'applicazione e i piani di sviluppo futuri.

Uso di connessioni DCOM

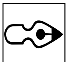
DCOM fornisce l'approccio più diretto alla comunicazione, in quanto non richiede ulteriori applicazioni di runtime sul server. Tuttavia, poiché DCOM non è incluso in Windows 95, le macchine client meno recenti potrebbero non averlo installato.

DCOM fornisce l'unico approccio che permette di usare i servizi di sicurezza quando si scrive un modulo dati transazionale. Questi servizi di sicurezza si basano sull'assegnazione di ruoli ai chiamanti degli oggetti transazionali. Quando si utilizza DCOM, DCOM identifica il chiamante come il sistema che effettua la chiamata all'application server (MTS o COM+). Pertanto, è possibile determinare in modo preciso il ruolo del chiamante. Quando si usano altri protocolli, tuttavia, c'è un eseguibile di runtime, diverso dall'application server, che riceve le chiamate del client. Questo eseguibile di runtime effettua le chiamate COM nell'application server per conto del client. A causa di ciò, è impossibile assegnare ruoli ai diversi clients: l'eseguibile di runtime risulta, in effetti, l'unico client. Per ulteriori informazioni sulla sicurezza e sugli oggetti transazionali, consultare il paragrafo "Sicurezza basata sul ruolo" a pagina 44-17.

Uso delle connessioni socket

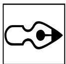
I socket TCP/IP permettono di creare client leggeri. Per esempio, se si stanno scrivendo applicazioni client per Web, non si può essere certi che i sistemi client supportino DCOM. I socket forniscono un minimo comune denominatore che sarà disponibile per la connessione all'application server. Per ulteriori informazioni sui socket, consultare il Capitolo 37, "Operazioni con i socket".

Anziché istanziare il modulo dati remoto direttamente dal client (come succede con DCOM), i socket usano una distinta applicazione sul server (ScktSrvr.exe), che accetta le richieste del client e istanzia l'application server usando COM. Il componente connection sul client e ScktSrvr.exe sul server sono responsabili dello smistamento delle *IAppServer*.



ScktSrvr.exe può essere eseguito come un'applicazione di servizio di NT. Per registrarlo come tale con Service manager, lo si deve avviare usando sulla riga comandi l'opzione -install. Per annullarne la registrazione, lo si deve avviare utilizzando sulla riga comandi l'opzione -uninstall.

Prima che si possa utilizzare una connessione socket, l'application server deve comunicare la sua disponibilità ai client utilizzando una connessione socket. Per impostazione predefinita, tutti i nuovi moduli dati remoti si registrano automaticamente mediante l'oggetto *TRemoteDataModuleRegistrar* nel metodo *UpdateRegistry* dell'oggetto di implementazione. È possibile impedire questa registrazione impostando a **false** la proprietà *EnableSocket* dell'oggetto.



Poiché i precedenti server non aggiungevano questa registrazione, è possibile disattivare il controllo sulla registrazione di un application server deselezionando la voce di menu Connections|Registered Objects Only di ScktSrvr.exe.

Quando si usano i socket, sul server non c'è alcuna protezione contro i guasti dei sistemi client prima che rilascino un riferimento alle interfacce sull'application server. Anche se si ha un minor traffico rispetto all'uso di DCOM (che invia messaggi

periodici per il mantenimento in attività), ciò può portare a un application server che non è in grado di rilasciare le sue risorse perché non sa che il client è spento.

Uso di connessioni

HTTP permette di creare client che possono comunicare con un application server protetto da un firewall. I messaggi HTTP forniscono un accesso controllato alle applicazioni interne, consentendo così una distribuzione estesa e in tutta sicurezza delle proprie applicazioni client. Analogamente alle connessioni socket, i messaggi HTTP forniscono un minimo comune denominatore che sarà disponibile per connettersi all'application server. Per maggiori informazioni su messaggi HTTP, consultare il Capitolo 32, "Creazione di applicazioni server per Internet".

Invece di istanziare direttamente il modulo dati remoto dal client (come accade con DCOM), le connessioni basate su HTTP utilizzano un'applicazione Web server sul server (httpsrvr.dll) che accetta le richieste dei client e istanzia l'application server utilizzando COM. Per questo motivo, sono chiamate anche connessioni Web. Il componente connessione sul client e httpsrvr.dll sul server sono responsabili del marshaling delle chiamate a *IAppServer*.

Le connessioni Web possono sfruttare la sicurezza SSL fornita da wininet.dll (una libreria di utilità per Internet che viene eseguita sul sistema client). Una volta configurato il server web sul sistema server in modo che richieda l'autenticazione, è possibile specificare il nome dell'utente e la password utilizzando le proprietà del componente connessione Web.

Come ulteriore misura di sicurezza, l'application server deve notificare la sua disponibilità ai client utilizzando una connessione Web. Per impostazione predefinita, tutti i nuovi moduli dati remoti si registrano automaticamente mediante l'oggetto *TRemoteDataModuleRegistrar* nel metodo *UpdateRegistry* dell'oggetto di implementazione. È possibile impedire questa registrazione impostando a **false** la proprietà *EnableWeb* dell'oggetto.

Le connessioni web possono trarre vantaggio dalla centralizzazione degli oggetti. Ciò consente il server di creare un sistema centralizzato limitato di istanze dell'application server, disponibili per le richieste dei client. Centralizzando gli application server, il server non utilizza risorse per il modulo dati e per la connessione al database se non quando sono necessarie. Per maggiori informazioni sulla centralizzazione di oggetti, consultare la sezione "Centralizzazione di moduli dati remoti" a pagina 29-8.

A differenza di altri componenti connection, se la connessione viene stabilita via HTTP non è possibile utilizzare callback.

Utilizzo di connessioni SOAP

SOAP è il protocollo che è alla base del supporto della VCL o della CLX per le applicazioni Web Service. SOAP smista le chiamate ai metodi utilizzando una codifica XML. Le connessioni SOAP utilizzano HTTP come protocollo di trasporto.

Le connessioni SOAP hanno il vantaggio di funzionare nelle applicazioni multipiattaforma perché sono supportati sia in Windows che in Linux. Poiché le

connessioni SOAP utilizzano HTTP, hanno gli stessi vantaggi delle connessioni Web: HTTP offre un minimo comun denominatore disponibile su tutti i client e i client possono comunicare con un application server protetto da un ”firewall”. Per ulteriori informazioni sull’utilizzo di SOAP per la distribuzione di applicazioni, vedere il Capitolo 36, “Utilizzo dei Web Services”.

Come nel caso delle conenssioni HTTP, se la connessione viene stabilita via SOAP non è possibile utilizzare callback. Inoltre le connessioni SOAP impongono il limite di un solo modulo dati remoto nell’application server.

Creazione di un’applicazione multi-tier

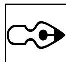
Le fasi fondamentali per la creazione di un’applicazione database multi-tier sono

1 Creare l’application server.

2 Registrare o installare l’application server.

3 Creare un’applicazione client.

La sequenza di creazione è importante. L’application server deve essere creato ed eseguito prima della creazione di un client. In fase di progettazione occorre collegarsi all’application server per collaudare il client. È possibile, ovviamente, creare un client senza specificare l’application server in fase di progettazione, e fornire solo il nome del server in fase di esecuzione. Però, così facendo, non sarà possibile capire se l’applicazione funziona come previsto nel codice in fase di progettazione, e non si potranno scegliere i server e i provider usando l’Object Inspector.



Se non si sta creando l’applicazione client sullo stesso sistema del server e non si sta utilizzando una connessione DCOM o socket, può essere necessario registrare o installare l’application server sul sistema client. Ciò associa il componente connection all’application server in fase di progettazione, consentendo così di scegliere i nomi del server e del provider da un elenco a discesa nell’Object Inspector. (Se si sta utilizzando una connessione Web, SOAP o socket, il componente connection recupera i nomi dei provider registrati dalla macchina server).

Creazione dell’application server

La creazione di un’application server è analoga alla creazione della maggior parte della applicazioni database. La differenza principale è che l’application server include un modulo dati remoto.

Per creare un’application server, fare quanto segue:

1 Avviare un nuovo progetto:

- Se come protocollo di trasporto si utilizza SOAP, il progetto dovrebbe essere una nuova applicazione Web Service. Scegliere File|New|Other, e sull pagina WebServices della finestra di dialogo New items, scegliere SOAP Server application.

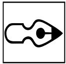
- Per qualsiasi altro protocollo di trasporto, è sufficiente scegliere File | New | Application.

Salvare il nuovo progetto.

2 Aggiungere un nuovo modulo dati remoto al progetto. Dal menu principale, scegliere File | New | Other, e sulle pagine MultiTier o WebServices della finestra di dialogo New Items selezionare

- **Remote Data Module** per la creazione di un Automation server COM a cui i client possano accedere usando DCOM, HTTP o socket.
- **Transactional Data Module** per la creazione di moduli dati remoti funzionanti in ambiente MTS o COM+. Le connessioni possono essere formate utilizzando DCOM, HTTP o i socket. Tuttavia, solo DCOM supporta i servizi per la sicurezza.
- **SOAP Server Data Module** se si sta creando un server SOAP in un'applicazione Web Service.

Per informazioni più dettagliate sulla configurazione di un modulo dati remoto, consultare "Impostazione del modulo dati remoto" a pagina 29-14.



Se si sceglie Remote Data Module oppure Transactional Data Module, il Wizard crea anche uno speciale oggetto COM Automation che contiene un riferimento al modulo dati remoto, utilizzato per cercare i provider. Tale oggetto è detto oggetto di implementazione. Nel caso di moduli dati SOAP, non v'e necessità di un oggetto di implementazione separato, in quanto il modulo dati implementa esso stesso l'interfaccia *IAppServerSOAP*.

3 Collocare gli appositi componenti dataset nel modulo dati e impostarli per accedere al database server.

4 Per ciascun dataset, collocare un componente *TDataSetProvider* sul modulo dati. Questo provider è necessario per eseguire il brokering delle richieste dei client e la creazione di pacchetti dati. Impostare la proprietà *DataSet* per ogni componente provider al nome del dataset a cui accedere. È possibile impostare ulteriori proprietà per il provider. Per informazioni più dettagliate sull'impostazione di un provider, consultare il Capitolo 28, "Uso di componenti provider".

Se si gestiscono dati provenieti da documenti XML, è possibile utilizzare un componente *TXMLTransformProvider* invece di un dataset e di un componente *TDataSetProvider*. Se si utilizza *TXMLTransformProvider*, impostare la proprietà *XMLDataFile* specificando il documento XML da cui provengono i dati e su cui applicare gli aggiornamenti.

5 Scrivere il codice dell' application server per implementare gli eventi, le regole condivise di gestione, la convalida condivisa dei dati e la sicurezza condivisa. Durante la stesura del codice, si potrebbe

- Estendere l'interfaccia dell'application server in modo da fornire all'applicazione client un ulteriore modo per chiamare il server. L'estensione dell'interfaccia dell'application server è descritta in "Estensione dell'interfaccia dell'application server" a pagina 29-17.

- Fornire un supporto per le transazioni in aggiunta alle transazioni automaticamente create al momento dell'applicazione degli aggiornamenti. Il supporto delle transazioni nella applicazioni database multi-tier è descritto in "Gestione delle transazioni nelle applicazioni multi-tiered" a pagina 29-18.
- Creare relazioni master/detail tra i dataset nell'application server. Le relazioni master/detail sono descritte in "Supporto delle relazioni master/detail" a pagina 29-19.
- Assicurasi che l'application server sia senza stato. La gestione di informazioni di stato è descritta in "Supporto delle informazioni di stato nei moduli dati remoti" a pagina 29-20.
- Dividere l'application server in più moduli dati remoti. L'utilizzo di più moduli dati remoti è descritto in "Uso di più moduli dati remoti" a pagina 29-21.

6 Salvare, compilare e registrare o installare l'application server. La registrazione di un application server è descritta in "Registrazione dell'application server" a pagina 29-22.

7 Se l'applicazione server non usa DCOM o SOAP, occorre installare il software di runtime che riceve i messaggi del client, istanzia il modulo dati remoto e smista le chiamate all'interfaccia.

- Per i socket TCP/IP questo è un'applicazione socket dispatcher, Scktsrvr.exe.
- Per le connessioni HTTP questo è httpsrvr.dll, una DLL ISAPI/NSAPI che deve essere installata con il server Web.

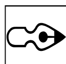
Impostazione del modulo dati remoto

Quando si crea il modulo dati remoto, occorre fornire alcune informazioni che indicano come rispondere alle richieste del client. Tali informazioni variano, a seconda del tipo di modulo dati remoto.

Configurazione del modulo dati remoto nel caso non sia transazionale

Per aggiungere un modulo dati remoto basato su COM alla applicazione senza includere attributi transazionali, scegliere File|New|Other e, dalla pagina Multitier della finestra di dialogo New Items, selezionare l'opzione Remote Data Module. Verrà visualizzato Remote Data Module wizard.

Bisogna fornire un nome di classe per il modulo dati remoto.Questo è il nome base di un discendente di *TCRemoteDataModule* che viene creato dall'applicazione. Esso è anche il nome base dell'interfaccia dell'application server. Ad esempio, se si usa come nome di classe *MyDataServer*, il wizard crea una nuova unit che dichiara *TMyDataServer*, un discendente di *TCRemoteDataModule*. Nell'header della unit, il Wizard dichiara inoltre una classe di implementazione (*TMyDataServerImpl*) che implementa *IMyDataServer*, un discendente di *IAppServer*.



Alla nuova interfaccia si possono aggiungere proprietà e metodi propri. Per ulteriori informazioni, consultare "Estensione dell'interfaccia dell'application server" a pagina 29-17.

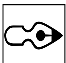
Occorre specificare il modello di threading nel Remote Data Module wizard. È possibile scegliere Single-threaded, Apartment-threaded, Free-threaded o Both.

- Se si sceglie Single-threaded, COM assicura che viene soddisfatta solo una richiesta del client alla volta. Non ci si deve preoccupare delle richieste dei client che interferiscono tra loro.
- Se si sceglie Apartment-threaded, COM assicura che qualsiasi istanza del modulo dati remoto soddisferà una richiesta alla volta. Quando si scrive del codice in una libreria Apartment-threaded, occorre prestare attenzione al rischio di conflitti tra thread se si usano variabili o oggetti globali non contenuti nel modulo dati remoto. Questo è il modello consigliato se si stanno utilizzando dataset BDE. (Si noti che, per gestire tutti gli aspetti relativi al threading sui dataset BDE, sarà necessario un componente session con la proprietà *AutoSessionName* impostata a **true**).
- Se si sceglie Free-threaded, l'applicazione può ricevere richieste simultanee di client su diversi thread. Bisogna accertarsi che l'applicazione sia thread-safe. Dal momento che più client possono accedere contemporaneamente al modulo dati remoto, occorre prestare attenzione ai dati dell'istanza (proprietà , oggetti contenuti, e così via) e alle variabili globali. Questo modello è consigliabile se si sta utilizzando un dataset ADO.
- Se si sceglie Both, la libreria funziona esattamente come quando si sceglie Free-threaded, ma con un'eccezione: tutte le callback (le chiamate alle interfacce client) vengono serializzate automaticamente.
- Se si sceglie Neutral, il modulo dati remoto può ricevere chiamate simultanee su thread separati, come nel modello Free-threaded, ma COM garantisce che non vi saranno mai due thread che accederanno contemporaneamente allo stesso metodo.

Configurazione di un modulo dati remoto transazionale

Per aggiungere un modulo dati remoto all'applicazione quando si utilizzerà MTS o COM+, scegliere il comando File|New|Other e selezionare il comando Transactional Data Module dalla pagina Multitier della finestra di dialogo New Items. Apparirà il wizard Transactional Data Module.

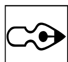
Bisogna fornire un nome di classe per il modulo dati remoto.Questo è il nome base di un discendente di *TCRemoteDataModule* che viene creato dall'applicazione. Esso è anche il nome base dell'interfaccia dell'application server. Ad esempio, se si usa come nome di classe *MyDataServer*, il wizard crea una nuova unit che dichiara *TMyDataServer*, un discendente di *TCRemoteDataModule*. Nell'header della unit, il wizard dichiara anche la classe di implementazione, (*TMyDataServerImpl*) che implementa sia *IMyDataServer* (un discendente di *IAppServer*) sia *IObjectControl* (che è richiesto da tutti gli oggetti transazionali). *TMyDataServerImpl* include un membro dati per l'interfaccia *IObjectContext*, che si potrà utilizzare per gestire transazioni, controllare la sicurezza, e così via.



Alla nuova interfaccia si possono aggiungere proprietà e metodi propri. Per ulteriori informazioni, consultare "Estensione dell'interfaccia dell'application server" a pagina 29-17.

Occorre specificare il modello di threading nel wizardTransactional Data Module. Scegliere tra Single, Apartment o Both.

- Se si sceglie Single, le richieste dei client vengono serializzate in modo che l'applicazione ne soddisfi una alla volta. Non ci si deve preoccupare delle richieste dei client che interferiscono tra loro.
- Se si sceglie Apartment, il sistema assicura che qualsiasi istanza del modulo dati remoto soddisfi una richiesta alla volta e che le chiamate usino sempre lo stesso thread. Occorre prestare attenzione al rischio di conflitti tra thread se si usano variabili o oggetti globali non contenuti nel modulo dati remoto. Anziché usare variabili globali, si può ricorrere allo Shared Property manager. Per maggiori informazioni sullo Shared Property manager, consultare il paragrafo "Shared property manager" a pagina 44-7.
- Se si sceglie Both, MTS chiama l'interfaccia dell'application server come quando si sceglie Apartment. Tuttavia, tutte le callback dirette alle applicazioni client vengono serializzate, cosicché non ci si deve preoccupare di quelle che interferiscono l'una con l'altra.



Il modello Apartment sotto MTS o COM+ è diverso dal modello corrispondente sotto DCOM.

Bisogna anche specificare gli attributi di transazione del modulo dati remoto. È possibile scegliere una delle seguenti opzioni:

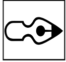
- Requires a transaction. Quando si seleziona questa opzione, ogni volta che un client usa l'interfaccia dell'application server, quella chiamata viene eseguita nel contesto di una transazione. Se il chiamante fornisce una transazione, non c'è alcuna necessità di crearne una nuova.
- Requires a new transaction. Quando si seleziona questa opzione, ogni volta che un client usa l'interfaccia dell'application server, viene creata automaticamente una nuova transazione per quella chiamata.
- Supports transactions. Quando si seleziona questa opzione, l'application server può essere usato nel contesto di una transazione, ma il chiamante deve fornire la transazione quando richiama l'interfaccia.
- Does not support transactions. Quando si seleziona questa opzione, l'application server non può essere usato nel contesto delle transazioni.

Configurazione di TSoapDataModule

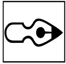
Per aggiungere all'applicazione un componente *TSoapDataModule*, scegliere File | New | Other e, dalla pagina WebService della finestra di dialogo New Items, selezionare l'opzione SOAP Data Module. Viene visualizzato SOAP data module wizard.

Bisogna fornire un nome di classe per il modulo dati SOAP. Questo è il nome di base di un discendente di *TSoapDataModule* che viene creato dall'applicazione. Ad esempio, se si usa come nome di classe *MyDataServer*, il wizard crea una nuova unit che dichiara *TMyDataServer*, un discendente di *TSoapDataModule*. Questa nuova classe eredita da *TSoapDataModule* un'implementazione di *IAppServer* e di *IAppServerSOAP*.

A differenza degli altri moduli dati remoti, i moduli dati SOAP non implementano un'interfaccia che discende da *IAppServer* o da *IAppServerSOAP*. Questo è dovuto alle differenze nel modo in cui le applicazioni Web Service smistano le chiamate in arrivo alle interfacce. Invece, utilizzando Add Web Service wizard è possibile alla propria applicazione nuove interfacce.



Per utilizzare *TSoapDataModule*, il nuovo modulo dati dovrebbe essere aggiunto a un'applicazione Web Service. L'interfaccia *IAppServerSOAP* è un'interfaccia richiamabile, registrata nel codice di inizializzazione della nuova unit. Ciò permette al componente invoker nel modulo Web principale di inoltrare tutte le chiamate in arrivo al modulo dati.



Se si desidera che l'application server risponda a client che sono stati scritti utilizzando Kylix 2 o Delphi 6 (prima della patch di aggiornamento 2), è necessario aggiungere del codice per registrare l'interfaccia *IAppServer*. Individuare il codice di avvio che registra *IAppServerSOAP*. Immediatamente dopo tale chiamata di registrazione, aggiungere una chiamata alla funzione globale *RegDefIAppServerInvClass*, la quale registra il modulo dati come implementazione di *IAppServer*.

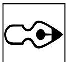
Estensione dell'interfaccia dell'application server

Le applicazioni client dei server basati su COM interagiscono con l'application server creando o connettendosi alla classe dell'implementazione creata da data module Wizard. Esse usano l'interfaccia del modulo dati remoto come base per tutte le comunicazioni con l'application server.

Se si utilizza un application server basato su COM, è possibile aggiungere elementi all'interfaccia della classe dell'implementazione in modo da fornire un supporto aggiuntivo alle applicazioni client. Questa interfaccia è un discendente di *IAppServer* e viene generata automaticamente dal wizard quando si crea il modulo dati remoto.

Per aggiungere elementi all'interfaccia del modulo dati remoto, usare l'editor delle librerie di tipi. Per maggiori informazioni sull'utilizzo dell'editor delle librerie di tipi, consultare il Capitolo 39, "Funzionamento delle librerie di tipi".

Quando si aggiungono elementi a un'interfaccia COM, le modifiche vengono aggiunte al codice sorgente della unit e al file della libreria di tipi (.TLB).



È necessario salvare esplicitamente il file .TLB scegliendo Refresh nel Type Library editor e salvando poi le modifiche dall'IDE.

Una volta aggiunta l'interfaccia della classe dell'implementazione, si devono individuare le proprietà e i metodi che sono stati aggiunti alla propria classe dell'implementazione. Quindi, si deve aggiungere il codice per portare a termine questa implementazione completando i corpi dei nuovi metodi.

Le applicazioni client chiamano le estensioni dell'interfaccia usando la proprietà *AppServer* del loro componente connection. Per ulteriori informazioni su come eseguire questa operazione, consultare "Chiamata delle interfacce del server" a pagina 29-29.

Aggiunta di callback all'interfaccia dell'application server

Si può consentire all'application server di chiamare l'applicazione client introducendo una callback. A questo scopo, l'applicazione client passa un'interfaccia a uno dei metodi dell'application server, e l'application server, in seguito, chiama questo metodo quando occorre. Comunque, se le estensioni all'interfaccia della classe implementation includono callback, non è possibile utilizzare una connessione basata su HTTP o su SOAP. *TWebConnection* e *TSoapConnection* non supportano le callback. Se si sta utilizzando una connessione basata su socket, le applicazioni client devono indicare se utilizzano o meno callback mediante l'impostazione della proprietà *SupportCallbacks*. Tutti gli altri tipi di connessione supportano automaticamente le callback.

Estensione dell'interfaccia transazionale di un application server

Quando si utilizzano le transazioni o l'attivazione just-in-time, occorre accertarsi che tutti i nuovi metodi chiamino il metodo *SetComplete* di IObjectContext per indicare che hanno terminato il proprio compito. Ciò consente di completare le transazioni e consente all'application server di essere disattivato.

Inoltre, i nuovi metodi non possono restituire alcun valore che consenta al client di comunicare direttamente con gli oggetti o con le interfacce sull'application server, a meno che non forniscano un riferimento sicuro. Se si sta utilizzando un modulo dati MTS senza stato, la dimenticanza dell'utilizzo di un riferimento sicuro può condurre a errore critici in quanto non è possibile garantire che il modulo dati remoto sia attivo. Per maggiori informazioni sui riferimenti sicuri, consultare "Passaggio di riferimenti a oggetti" a pagina 44-27.

Gestione delle transazioni nelle applicazioni multi-tiered

Quando le applicazioni client applicano gli aggiornamenti all'application server, il componente provider effettua automaticamente il wrap del processo di applicazione degli aggiornamenti e di risoluzione degli errori in una transazione. Questa transazione viene inoltrata se il numero di record problematici non supera il valore *MaxErrors* specificato come argomento per il metodo *ApplyUpdates*. Altrimenti, tutto viene annullato.

Inoltre, si può aggiungere il supporto per le transazioni all'applicazione server aggiungendo ricorrendo a un componente di connessione al database oppure gestendo direttamente la transazione tramite l'invio di comandi SQL al server di database. Questo funziona nello stesso modo in cui si gestiscono le transazioni in un'applicazione two-tier. Per ulteriori informazioni su questo tipo di controllo della transazione, consultare "Gestione delle transazioni" a pagina 21-6.

Se si usa un modulo dati transazionale, è possibile ampliare il supporto transazionale usando le transazioni MTS o COM+. Queste possono includere qualsiasi logica di gestione sull'application server, non solo l'accesso al database. Inoltre, poiché supportano gli invii in due fasi, le transazioni possono interessare più database.

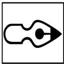
Solamente i componenti di accesso ai dati basati su BDE e ADO supportano l'invio a due fasi. Non utilizzare componenti InterbaseExpress o dbExpress se si desidera avere transazioni che coinvolgono più database.



Quando si utilizza BDE, l'invio in due fasi viene implementato completamente solo sui database Oracle7 e MS-SQL. Se la transazione riguarda più database, alcuni dei quali sono server remoti diversi da Oracle7 o MS-SQL, la transazione corre il rischio di una riuscita solo parziale. All'interno di un qualsiasi database, tuttavia, ci sarà sempre un supporto transazionale.

Per impostazione predefinita, tutte le chiamate di *IAppServer* a un modulo dati transazionale sono transazionali. È necessario solamente impostare l'attributo della transazione del modulo dati per indicare che deve partecipare alle transazioni. Inoltre, è possibile estendere l'interfaccia dell'application server in modo da includere le chiamate ai metodi che incapsulano le transazioni definite dal programmatore.

Se l'attributo della transazione indica che l'application server richiede una transazione, ogni volta che un client chiama un metodo della propria interfaccia, viene inglobato automaticamente in una transazione. Tutte le chiamate del client all'application server vengono quindi elencate in quella transazione finché non si indica che è completata. Queste chiamate o riescono completamente oppure vengono annullate.



Non si devono abbinare le transazioni MTS o COM+ alle transazioni esplicite create da un componente di connessione al database o mediante un comando SQL esplicito. Quando il modulo dati transazionale viene elencato in una transazione, elenca automaticamente anche tutte le chiamate al database contenute nella transazione.

Per ulteriori informazioni sull'uso delle transazioni MTS e COM+, consultare il paragrafo "Supporto delle transazioni MTS e COM+" a pagina 44-10.

Supporto delle relazioni master/detail

È possibile creare una relazione master/detail tra i dataset client in un'applicazione client nello stesso modo utilizzato per impostarle quando si utilizza un qualsiasi dataset di tipo tabella. Per ulteriori informazioni sull'impostazione di relazioni master/detail, consultare "Creazione di relazioni master/detail" a pagina 22-36.

Tuttavia, questo approccio presenta due importanti svantaggi:

- La tabella detail deve recuperare e memorizzare tutti i suoi record dall'application server anche se usa solo un set detail alla volta. (Questo problema può essere mitigato mediante l'uso di parametri. Per ulteriori informazioni, consultare "Limitazione dei record mediante parametri" a pagina 27-31.)
- È molto difficile applicare gli aggiornamenti, perché i dataset client li applicano a livello di dataset e perché gli aggiornamenti master/detail riguardano più dataset. Anche in un ambiente two-tier, dove si può usare il componente di connessione al database per applicare gli aggiornamenti a più tabelle con un'unica transazione, l'applicazione degli aggiornamenti nelle schede master/detail è complessa.

Nelle applicazioni multi-tier questi problemi possono essere evitati usando le tabelle annidate per rappresentare la relazione master/detail. A questo scopo, durante la fase di fornitura dai dataset, occorre impostare una relazione master/detail tra i dataset sull'application server. Quindi, si deve impostare la proprietà *DataSet* del componente provider alla tabella master. Per utilizzare tabelle annidate per rappresentare le relazioni di master/detail durante la fornitura a partire da documenti XML, utilizzare un file di trasformazione che definisce i set di dettaglio annidati.

Quando i client chiamano il metodo *GetRecords* del provider, questi include automaticamente i dataset di dettaglio come campo *DataSet* nei record del pacchetto dati. Quando i client chiamano il metodo *ApplyUpdates* del provider, questo gestisce automaticamente l'applicazione degli aggiornamenti nel corretto ordine.

Supporto delle informazioni di stato nei moduli dati remoti

L'interfaccia *IAppServer*, che i client dataset utilizzano per comunicare con i provider sull'application server, è quasi sempre senza stato. Se un'applicazione è senza stato, non "ricorda" niente di ciò che è successo nelle precedenti chiamate da parte del client. Questa qualità di essere senza stato risulta utile se si vogliono centralizzare le connessioni al database in un modulo dati transazionale, in quanto l'application server non ha la necessità di riconoscere distintamente le informazioni persistenti, come, ad esempio, il record corrente, tra una connessioni al database e la successiva. Analogamente, questa qualità di essere senza stato diventa importante quando si condividono istanze di moduli dati remoti tra diversi client, come accade con l'attivazione just-in-time o con la centralizzazione degli oggetti. I moduli dati SOAP devono essere senza stato.

A volte, tuttavia, potrebbe essere necessario conservare le informazioni di stato tra una chiamata all'application server e l'altra. Per esempio, quando si richiedono dati utilizzando il recupero incrementale, il provider sull'application server deve "ricordare" le informazioni delle chiamate precedenti (qual è il record corrente).

Prima e dopo ogni chiamata all'interfaccia *IAppServer* effettuata dal dataset client (AS_*ApplyUpdates*, AS_*Execute*, AS_*GetParams*, AS_GetRecordso AS_*RowRequest*), l'application server riceve un evento in cui può trasmettere o recuperare informazioni di stato personalizzate. Analogamente, i provider, prima e dopo ogni risposta a queste chiamate generate dal client, ricevono eventi in cui possono reperire o trasmettere informazioni di stato personalizzate. Utilizzando questo meccanismo, è possibile comunicare informazioni di stato persistenti tra le applicazioni client e l'application server, anche se l'application server è senza stato.

Ad esempio, si consideri un dataset che rappresenta la seguente query parametrizzata:

```
SELECT * from CUSTOMER WHERE CUST_NO > :MinVal ORDER BY CUST_NO
```

Per consentire il prelievo incrementale in un application server senza stato si può fare quanto segue:

- Quando il provider assembla un set di record in un pacchetto dati, annota il valore di CUST_NO sull'ultimo record nel pacchetto:

```
TRemoteDataModule1::DataSetProvider1GetData(TObject *Sender, TCustomClientDataSet *DataSet)
{
  DataSet->Last(); // move to the last record
  TComponent *pProvider = dynamic_cast<TComponent *>(Sender);
  pProvider->Tag = DataSet->FieldValues["CUST_NO"];
}
```

- Il provider invia questo ultimo valore CUST_NO al client dopo avere mandato il pacchetto dati:

```
TRemoteDataModule1::DataSetProvider1AfterGetRecords(TObject *Sender, OleVariant &OwnerData)
{
  TComponent *pProvider = dynamic_cast<TComponent *>(Sender);
  OwnerData = pProvider->Tag;
}
```

- Sul client, il dataset client salva questo ultimo valore di CUST_NO:

```
TDataModule1::ClientDataSet1AfterGetRecords(TObject *Sender, OleVariant &OwnerData)
{
  TComponent *pDS = dynamic_cast<TComponent *>(Sender);
  pDS->Tag = OwnerData;
}
```

- Prima di prelevare un pacchetto dati, il client invia l'ultimo valore di CUST_NO che ha ricevuto:

```
TDataModule1::ClientDataSet1BeforeGetRecords(TObject *Sender, OleVariant &OwnerData)
{
  TClientDataSet *pDS = dynamic_cast<TClientDataSet *>(Sender);
  if (!pDS->Active)
    return;
  OwnerData = pDS->Tag;
}
```

- Infine, sul server, il provider utilizza l'ultimo CUST_NO inviato come valore minimo nella query:

```
TRemoteDataModule1::DataSetProvider1BeforeGetRecords(TObject *Sender, OleVariant &OwnerData)
{
  if (!VarIsEmpty(OwnerData))
  {
    TDataSetProvider *pProv = dynamic_cast<TDataSetProvider *>(Sender);
    TSQLDataSet *pDS = (dynamic_cast<TSQLDataSet *>(pProv->DataSet);
    pDS->Params->ParamValues["MinVal"] = OwnerData;
    pDS->Refresh(); // force the query to reexecute
  }
}
```

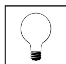
Uso di più moduli dati remoti

Potrebbe essere necessario strutturare l'application server in modo che utilizzi più moduli dati remoti. L'utilizzo di più moduli dati remoti permette di suddividere il codice, organizzando un grosso application server in più unit, in cui ogni unit è sufficientemente indipendente.

Benché sia sempre possibile creare più moduli dati remoti sull'application server che operano in modo indipendente, un particolare componente connection sulla pagina DataSnap della Component Palette dà la possibilità di usare un modello in cui si ha un modulo dati remoto ”genitore” principale che indirizza le connessioni dai client ad altri moduli dati remoti ”figlio”. Questo modello richiede che si utilizzi un application server basato su COM.

Per creare il modulo dati remoto genitore, è necessario ampliare la sua interfaccia *IAppServer*, aggiungendo proprietà che espongono le interfacce dei moduli dati remoti figlio. Vale a dire, per ogni modulo dati remoto figlio, si deve aggiungere una proprietà all'interfaccia del modulo dati genitore il cui valore è l'interfaccia *IAppServer* del modulo dati figlio.

Per ulteriori informazioni sull'estensione dell'interfaccia dell'parent remote, consultare "Estensione dell'interfaccia dell'application server" a pagina 29-17.



Potrebbe anche esere necessario ampliare l'interfaccia di ogni modulo dati figlio, esponendo l'interfaccia del modulo dati genitore o le interfacce degli altri moduli dati figlio. Ciò permette ai diversi moduli dati nell'application server di comunicare l'uno l'altro più liberamente.

Una volta aggiunte al modulo dati remoto principale le proprietà che rappresentano i moduli dati remoti figlio, le applicazioni client devono più stabilire connessioni separate con ogni modulo dati remoto sull'application server. Esse condividono invece una singola connessione al modulo dati remoto genitore, il quale al sua volta indirizza messaggi ai moduli dati ”figlio”. Poiché ogni applicazione client utilizza la stessa connessione per ogni modulo dati remoto, i moduli dati remoti possono condividere una singola connessione database, risparmiando risorse. Per informazioni sulle modalità di condivisione di una singola connessione da parte delle applicazioni figlio, vedere "Connessione a un application server che utilizza più moduli dati" a pagina 29-30.

Registrazione dell'application server

Prima che le applicazioni client possano individuare e utilizzare un application server, deve essere registrate o installato.

- Se l'application server usa come protocollo di comunicazione DCOM, HTTP o i socket, si comporta come un Automation server e deve essere registrato come un qualunque altro server COM. Per informazioni sulla registrazione di un server COM, consultare "Registrazione di un oggetto COM" a pagina 41-17.
- Se si utilizza un modulo dati transazionale, non si registra l'application server. Lo si installerà , invece, insieme con MTS o COM+. Per maggiori informazioni sull'installazione di oggetti transazionali, consultare il paragrafo "Installazione di oggetti transazionali" a pagina 44-29.
- Se l'application server utilizza SOAP, l'applicazione deve essere un'applicazione Web Service. Come tale, deve essere registrata con il Web Server, in modo che riceva i messaggi HTTP in arrivo. Inoltre, è possibile pubblicare un documento WSDL che descrive le interfacce richiamabili nell'applicazione. Per informazioni

sull'esportazione di un documento WSDL per un'applicazione Web Service, vedere “Generazione di documenti WSDL per un'applicazione Web Service” a pagina 36-16.

Creazione dell'applicazione client

Per molti aspetti la creazione di un'applicazione client multi-tier è simile alla creazione un client two-tier che utilizza un dataset client per memorizzare in cache gli aggiornamenti. La princpale differenza consiste nel fatto che un'applicazione multi-tier utilizza un componente connection per stabilire un collegamento con l' application server.

Per creare un'applicazione client multi-tier, si deve avviare un nuovo progetto e eseguire queste operazioni:

1 Aggiungere un nuovo modulo dati al progetto.
2 Collocare un componente connection nel modulo dati. Il tipo di componente connessione che si aggiunge dipende dal protocollo di comunicazione che si vuole utilizzare. Per i dettagli, consultare “Struttura dell'applicazione client” a pagina 29-4.
3 Impostare le proprietà del componente connessione in modo da specificare l'application server con cui si dovrebbe istituire una connessione. Per ulteriori informazioni sulla configurazione dei componenti connessione, consultare la sezione “Connessione all'application server” a pagina 29-24.
4 Impostare le altre proprietà del componente connection come richiesto per l'applicazione. Per esempio, si può impostare la proprietà *ObjectBroker* per consentire al componente connection di scegliere dinamicamente da più server. Per maggiori informazioni circa l'utilizzo di componenti connection, consultare “Gestione delle connessioni al server” a pagina 29-27
5 Collocare tutti i componenti *TClientDataSet* necessari sul modulo dati, e impostare la proprietà *RemoteServer* per ciascun componente al nome del componente connection collocato al Punto 2. Per una completa introduzione ai dataset client, consultare il Capitolo 27, “Utilizzo dei dataset client”.
6 Impostare la proprietà *ProviderName* di ciascun componente *TClientDataSet*. Se il componente connection viene collegato all'application server in fase di progettazione, è possibile scegliere i provider disponibili dell'application server dall'elenco a discesa della proprietà *ProviderName*.
7 Continuare esattamente come se si stesse creando una qualsiasi altra applicazione database. Vi sono alcune funzioni aggiuntive disponibili per client delle applicazioni multi-tier:
   - L'applicazione potrebbe dover chiamare direttamente l'application server. Queste operazioni sono descritte in “Chiamata delle interfacce del server” a pagina 29-29.
   - Molto probabilmente si vorranno utilizzare le particolari funzionalità dei dataset client che supportano la loro interazione con i componenti provider.

Queste funzionalità sono descritte nella sezione "Uso di un dataset client con un provider di dati" a pagina 27-26.

Connessione all'application server

Per stabilire e mantenere una connessione a un application server, un'applicazione client usa uno o più componenti connection. Questi componenti si trovano sulle pagine DataSnap o WebServices della Component palette.

Un componente connection può essere utilizzato per

- Identificare il protocollo per la comunicazione con l'application server. Ogni tipo di componente connection rappresenta un protocollo di comunicazione differente. Per informazioni dettagliate sui vantaggi e sui vincoli dei protocolli disponibili, consultare il paragrafo "Scelta di un protocollo di connessione" a pagina 29-9.
- Indicare come individuare la macchina server. I dettagli per l'identificazione della macchina server variano a seconda del protocollo.
- Identificare l'application server sulla macchina server.
- Se non si sta utilizzando SOAP, identificare il server usando la proprietà *ServerName* o *ServerGUID*. *ServerName* identifica il nome di base della classe specificata al momento della creazione del modulo dati remoto sull'application server. Per informazioni dettagliate su come viene specificato questo valore sul server, consultare il paragrafo "Impostazione del modulo dati remoto" a pagina 29-14. Se il server viene registrato o installato sulla macchina client, o se il componente connection viene collegato alla macchina server, è possibile impostare la proprietà *ServerName* in fase di progettazione scegliendo da un elenco a discesa nell'Object Inspector. *ServerGUID* specifica il GUID dell'interfaccia del modulo dati remoto. È possibile consultare questo valore usando il Type Library editor.

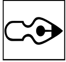
  Se si utilizza SOAP, il server è identificato nello *URL* che si usa per individuare la macchina server. Seguire le istruzioni riportate in "Specifica di una connessione con SOAP" a pagina 29-26.
- Gestire le connessioni al server. I componenti connection possono essere usati per creare o abbandonare le connessioni e per chiamare le interfacce dell'application server.

Normalmente, l'application server è su una macchina diversa dall'applicazione client, ma anche se il server risiede sulla stessa macchina dell'applicazione client (per esempio, durante la costruzione e la verifica dell'intera applicazione multi-tier), è sempre possibile usare il componente connection per identificare l'application server per nome, specificare una macchina server e usare l'interfaccia dell'application server.

Specifica di una connessione con DCOM

Quando si usa DCOM per comunicare con l'application server, le applicazioni client includono un componente *TDCOMConnection* per la connessione all'application server. *TDCOMConnection* usa la proprietà *ComputerName* per identificare la macchina su cui risiede il server.

Quando *ComputerName* è vuota, il componente connection DCOM assume che l'application server risiede sulla macchina client o che l'application server ha un elemento nel registro di sistema. Se, quando si usa DCOM, non si fornisce un elemento del registro di sistema per l'application server sul client, e il server risiede su una macchina diversa dal client, bisogna indicare *ComputerName*.



Anche quando c'è un elemento nel registro di sistema per l'application server, si può specificare *ComputerName* per ridefinirlo. Ciò può essere particolarmente utile durante lo sviluppo, il collaudo e il debug.

Se ci sono più server da cui l'applicazione client può scegliere, è possibile usare la proprietà *ObjectBroker* anziché specificare un valore per *ComputerName*. Per ulteriori informazioni, consultare il paragrafo "Brokering delle connessioni" a pagina 29-27.

Se si fornisce il nome di un computer host o di un server inesistente, quando si cerca di aprire la connessione, il componente connection DCOM solleva un'eccezione.

Specifica di una connessione con i socket

È possibile stabilire una connessione all'application server usando i socket su qualsiasi macchina che abbia un indirizzo TCP/IP. Questo metodo ha il vantaggio di essere applicabile a più macchine, ma non fornisce la possibilità di usare tutti i protocolli di sicurezza. Quando si usano i socket, per collegarsi all'application server bisogna includere un componente *TSocketConnection*.

TSocketConnection identifica la macchina server usando l'indirizzo IP o il nome dell'host del sistema server e il numero di porta del programma socket dispatcher (Scktsrvr.exe) che viene eseguito sulla macchina server. Per ulteriori informazioni sugli indirizzi IP e sui valori della porta, consultare "Descrizione dei socket" a pagina 37-3.

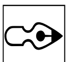
Tre proprietà di *TSocketConnection* specificano queste informazioni:

- *Address* specifica l'indirizzo IP del server.
- *Host* specifica il nome dell'host del server.
- *Port* specifica il numero di porta del programma socket dispatcher sull'application server.

Address and *Host* si escludono a vicenda. L'impostazione di uno annulla il valore dell'altra. Per informazioni su quale dei due usare, consultare "Descrizione dell'host" a pagina 37-4.

Se ci sono più server da cui l'applicazione client può scegliere, è possibile usare la proprietà *ObjectBroker* anziché specificare un valore per *Address* o *Host*. Per ulteriori informazioni, consultare il paragrafo "Brokering delle connessioni" a pagina 29-27.

Per impostazione predefinita, il valore di *Port* è 211, che è il numero di porta predefinito del programma socket dispatcher che inoltra messaggi in arrivo all'application server. Se il socket dispatcher è stato configurato per usare un'altra porta, occorre impostare la proprietà *Port* in modo che corrisponda a quel valore.



È possibile configurare la porta del socket dispatcher mentre è in esecuzione facendo clic sull'icona Borland Socket Server nella barra di stato e scegliendo Properties.

Sebbene le connessioni socket non consentano l'utilizzo di protocolli di sicurezza, è possibile personalizzare la connessione socket aggiungendo funzionalità di crittografia personalizzate. Per raggiungere questo risultato

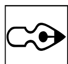
1. Creare un oggetto COM che supporti l'interfaccia *IDataIntercept*. Questa è un'interfaccia per la crittografia e la decifrazione dei dati.
2. Registrare il nuovo server COM sulla macchina client.
3. Impostare la proprietà *InterceptName* o *InterceptGUID* del componente connessione socket per specificare questo oggetto COM.
4. Infine, fare clic destro sull'icona Borland Socket Server nella barra delle applicazioni, selezionare Properties e sulla pagina delle proprietà impostare Intercept Name o Intercept GUID al ProgId o al GUID dell'interceptor.

Tale meccanismo può essere utilizzato anche per la compressione e la decompressione dei dati.

Specifica di una connessione con HTTP

È possibile stabilire una connessione all'application server usando HTTP su qualsiasi macchina che abbia un indirizzo TCP/IP. A differenza dei socket, tuttavia, HTTP consente di sfruttare la sicurezza SSL e di comunicare con un server protetto da un firewall. Quando si usano gli HTTP, per collegarsi all'application server bisogna includere un componente *TWebConnection*.

Il componente connessione Web stabilisce una connessione con l'applicazione Web server (httpsrvr.dll), che a sua volta comunica con l'application server. *TWebConnection* individua httpsrvr.dll utilizzando uno URL (Uniform Resource Locator). L'URL specifica il protocollo (http oppure, se si sta utilizzando la sicurezza SSL, https), il nome di host della macchina sui sui sono in secuzione il server web e httpsrvr.dll, e il percorso per l'applicazione Web server (httpsrvr.dll). Specificare questo valore utilizzando la proprietà *URL*.



Quando si utilizza un componente *TWebConnection*, è necessario che wininet.dll sia installato sulla macchina client. Se è stato installato IE3, o una versione successiva, wininet.dll è presente nella directory System di Windows.

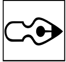
Se il server Web richiede l'autenticazione, o si sta utilizzando un proxy server che richiede l'autenticazione, occorre impostare i valori delle proprietà *UserName* e *Password* in modo che il componente connessione possa effettuare il login.

Se l'applicazione client può scegliere tra più server, invece di specificare un valore per la proprietà *URL* si può utilizzare la proprietà *ObjectBroker*. Per ulteriori informazioni, consultare il paragrafo "Brokering delle connessioni" a pagina 29-27.

Specifica di una connessione con SOAP

È possibile stabilire una connessione a un application server SOAP utilizzando il componente *TSoapConnection*. *TSoapConnection* è molto simile a *TWebConnection* perché come protocollo di trasporto utilizza anch'esso HTTP. Perciò, è possibile utilizzare *TSoapConnection* da qualsiasi macchina dotata di un indirizzo TCP/IP e sfruttare la sicurezza SSL per comunicare con un server protetto da un firewall.

Il componente SOAP connection stabilisce una connessione con un Web Service che implementa l'interfaccia *IAppServerSOAP* e *IAppServer*. (La proprietà *UseSOAPAdapter* specifica qual è l'interfaccia prevista che dovrebbe essere supportata dal server). Se il server implementa l'interfaccia *IAppServerSOAP*, *TSoapConnection* converte tale interfaccia in un'interfaccia *IAppServer* per i dataset client. *TSoapConnection* individua questa applicazione Web server utilizzando uno URL (Uniform Resource Locator). L'*URL* specifica il protocollo (http oppure, se si sta utilizzando la sicurezza SSL, https), il nome di host della macchina sui sui è in secuzione il server web, il nome dell'applicazione Web Service, e un percorso che coincide con il nome di percorso del *THTTPSoapDispatcher* sull'application server. Specificare questo valore utilizzando la proprietà *URL*.



Quando si utilizza un componente *TSoapConnection*, è necessario che wininet.dll sia installato sulla macchina client. Se è stato installato IE3, o una versione successiva, wininet.dll è presente nella directory System di Windows.

Se il server Web richiede l'autenticazione, o si sta utilizzando un proxy server che richiede l'autenticazione, occorre impostare i valori delle proprietà *UserName* e *Password* in modo che il componente connessione possa effettuare il login.

Brokering delle connessioni

Se ci sono più server basati su COM da cui l'applicazione client può scegliere, si può usare un Object Broker per individuare un sistema server disponibile. L'Object Broker conserva un elenco di server da cui il componente connection può scegliere. Quando il componente connection ha bisogno di collegarsi a un application server, chiede all'Object Broker il nome di un computer (o un indirizzo IP, un nome di host, o *URL*). Il broker fornisce un nome e il componente connection stabilisce una connessione. Se il nome fornito non va bene (per esempio, se il server è spento), il broker fornisce un altro nome, e così via, finché non viene stabilita una connessione.

Una volata che il componente connection ha stabilito una connessione con un nome fornito dal broker, salva quel nome come valore della relativa proprietà (*ComputerName*, *Address*, *Host*, *RemoteHost*, o *URL*). Se in seguito il componente connection chiude la connessione ed ha poi bisogno di riaprirla, cerca di usare il valore di questa proprietà , e richiede un nuovo nome dal broker solo se la connessione non avviene.

Per usare un Object Broker, si deve specificare la proprietà *ObjectBroker* del componente connection. Quando la proprietà *ObjectBroker* è impostata, il componente connection non salva il valore di *ComputerName*, *Address*, *Host*, *RemoteHost*, o *URL* su disco.

Gestione delle connessioni al server

Lo scopo principale dei componenti connection è di individuare e di effettuare la connessione all'application server. Poiché gestiscono le connessioni al server, questi componenti possono anche essere utilizzati per chiamare i metodi dell'interfaccia dell'application server.

Connessione al server

Per individuare e collegarsi all'application server, prima è necessario impostare le proprietà del componente connection per identificare l'application server. Questo processo viene descritto nella sezione "Connessione all'application server" a pagina 29-24. Prima di aprire la connessione, tutti i dataset client che usano il componente connection per comunicare con l' application server dovranno indicare questo fatto impostando la loro proprietà *RemoteServer* per specificare il componente connection.

La connessione viene aperta automaticamente quando i dataset client tentano di accedere all' application server. Per esempio, l'impostazione della proprietà *Active* del dataset client a **true** apre la connessione, sempre che la proprietà *RemoteServer* sia stata impostata.

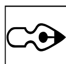
Se non si collegano i dataset client al componente connection, è possibile aprire la connessione impostando la proprietà *Connected* del componente connection a **true**.

Prima di stabilire una connessione a un application server, il componente connection genera un evento *BeforeConnect*. È possibile eseguire delle azione specifiche per collegare un gestore *BeforeConnect* tramite programma. Dopo aver stabilito la connessione, il componente connection genera un evento *AfterConnect* per tutte le azioni speciali.

Abbandono o modifica di una connessione a un server

Un componente connection abbandona una connessione all'application server quando

- Si imposta la proprietà *Connected* a **false**.
- Si libera il componente connection. Un oggetto connection viene liberato automaticamente quando un utente chiude l'applicazione client.
- Si modifica una delle proprietà che identificano l'application server (*ServerName*, *ServerGUID*, *ComputerName*, e così via). La modifica di queste proprietà consente di passare, in fase di esecuzione, da un application server disponibile all'altro. Il componente connection abbandona la connessione attiva e ne stabilisce una nuova.



Anziché ricorrere a un singolo componente connection per passare da un application server disponibile all'altro, un'applicazione client può utilizzare più componenti connection, ciascuno dei quali è collegato a un application server differente.

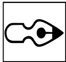
Prima di abbandonare una connessione, il componente connection chiama automaticamente il suo gestore di evento *BeforeDisconnect*, se ne è stato fornito uno. Se si vuole eseguire una qualsiasi particolare azione prima di annullare la connessione, scrivere un gestore *BeforeDisconnect*. Analogamente, dopo aver abbandonato la connessione, viene chiamato il gestore di evento *AfterDisconnect*. Se si vuole eseguire una qualsiasi particolare azione dopo aver annullato la connessione, scrivere un gestore *AfterDisconnect*.

Chiamata delle interfacce del server

Non è necessario che le applicazioni chiamino direttamente l'interfaccia *IAppServer* perché le chiamate opportune vengono generate automaticamente quando si usano le proprietà e i metodi del dataset client. Tuttavia, benché non sia necessario operare direttamente con l'interfaccia *IAppServer*, all'interfaccia dell'application server potrebbero essere state aggiunte delle estensioni personalizzate se non si sta utilizzando SOAP. Quando si estende l'interfaccia dell'application server, è necessario disporre di un modo per chiamare queste estensioni usando la connessione creata dal componente connection. È possibile compiere questa operazione usando la proprietà *AppServer* del componente connection. Per ulteriori informazioni sull'estensione dell'interfaccia dell'application server, consultare "Estensione dell'interfaccia dell'application server" a pagina 29-17.

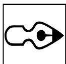
AppServer è un tipo Variant che rappresenta l'interfaccia dell'application server. Per chiamare questa interfaccia, è necessario ottenere da questo Variant un'interfaccia dispatch. L'interfaccia dispatch ha lo stesso nome dell'interfaccia creata al momento della creazione del modulo dati remoto, a cui viene aggiunto la stringa "Disp". Pertanto, se il modulo dati remoto si chiama MyAppServer, è possibile utilizzare *AppServer* per chiamarne la sua interfaccia nel seguente modo:

```
IDispatch* disp = (IDispatch*)(MyConnection->AppServer)
IMyAppServerDisp TempInterface( (IMyAppServer*)disp);
TempInterface.SpecialMethod(x,y);
```



L'interfaccia dispatch è dichiarata nel file _TLB.h generato dal Type Library editor.

Se si sta utilizzando SOAP, non è possibile usare la proprietà *AppServer*. È necessario invece utilizzare un oggetto interfacciato in remoto (*THTTPRio*) ed effettuare chiamate con binding anticipato. Come con tutte le chiamate con binding anticipato, l'applicazione client deve conoscere la dichiarazione dell'interfaccia dell'application server al momento della compilazione. È possibile aggiungere ciò all'applicazione client referenziando un documento WSDL che descrive l'interfaccia che si desidera chiamare. Notare che nel caso dei server SOAP, questa interfaccia è completamente separata dall'interfaccia del modulo dati SOAP. Per informazioni sull'importazione di un documento WSDL che descrive l'interfaccia del server, vedere "Importazione di documenti WSDL" a pagina 36-17.



La unit che dichiara l'interfaccia del server deve registrarla anche con il registro delle chiamata. Per i dettagli su come registrare le interfacce richiamabili, vedere "Le interfacce richiamabili" a pagina 36-2.

Una volta che si è importato un documento WSDL per generare una unit che dichiara e registra l'interfaccia, creare un'istanza di *THTTPRio* per l'interfaccia desiderata:

```
THTTPRio *X = new THTTPRio(NULL);
```

Quindi, assegnare all'oggetto interfacciato in remoto l'*URL* utilizzato dal componente connection, aggiungendo il nome dell'interfaccia che si desidera chiamare:

```
X->URL = SoapConnection1.URL + "IMyInterface";
```

Ora, è possibile utilizzare il metodo QueryInterface per ottenere un'interfaccia per chiamare i metodi del server:

```
InterfaceVariable = X->QueryInterface(IMyInterfaceIntf);
if (InterfaceVariable)
{
  InterfaceVariable->SpecialMethod(a,b);
}
```

Notare che la chiamata a *QueryInterface* accetta come argomento per l'interfaccia richiamabile il wrapper *DelphiInterface* invece della stessa interfaccia richiamabile.

Connessione a un application server che utilizza più moduli dati

Se un'application server basato su COM utilizza un modulo dati remoto "genitore" principale e numerosi moduli dati remoti figlio, come descritto in "Uso di più moduli dati remoti" a pagina 29-21, è necessario utilizzare un componente connection separato per ogni modulo dati remoto sull'application server. Ogni componente connection rappresenta la connessione a un singolo modulo dati remoto.

Benché sia possibile che l'applicazione client stabilisca connessioni indipendenti con ciascun modulo dati remoto sull'application server, risulta più efficiente utilizzare una singola connessione all'application server condivisa però da tutti i componenti connection. Vale a dire, si aggiunge un singolo componente connection che si collega al modulo dati remoto "principale" sull'application server e quindi, per ogni modulo dati remoto "figlio", si aggiunge un ulteriore componente che condivide la connessione al modulo dati remoto principale.

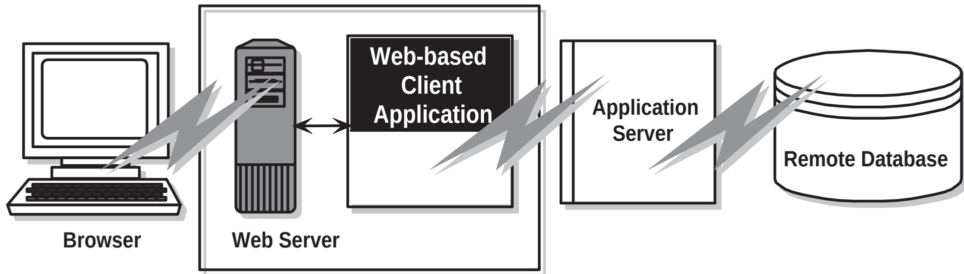
1 Per la connessione al modulo dati remoto principale, aggiungere e configurare un componente connection come descritto in "Connessione all'application server" a pagina 29-24. L'unica limitazione è che è impossibile utilizzare una connessione SOAP.
2 Per ogni modulo dati remoto figlio, utilizzare un componente *TSharedConnection*.
    - Impostare la sua proprietà *ParentConnection* al componente connection incluso passo 1. Il componente *TSharedConnection* condivide la connessione che questa connessione principale stabilisce.
    - Impostare la sua proprietà *ChildName* al nome della proprietà sull'interfaccia del modulo dati remoto principale che espone l'interfaccia del modulo dati remoto figlio chiesto.

Quando si assegna il componente *TSharedConnection* messo nel passo 2 come il valore un dataset client è proprietà *RemoteServer*, essa lavori come se si utilizzava una connessione completamente indipendente al modulo dati remoto figlio. Tuttavia, il componente *TSharedConnection* utilizza la connessione stabilita dal componente messo nel passo 1.

Scrittura di applicazioni client basate su Web

Se si vogliono creare client basati su Web per un'applicazione database multi-tier, si deve sostituire il tier client con una speciale applicazioni Web che agisca simultaneamente sia come un client dell'application server sia come un'applicazione Web server installata con un server web sulla stessa macchina. Questa architettura è illustrata nella Figura 29.1.

Figura 29.1 Applicazione database multi-tier basata su Web



La generazione dell'applicazione Web può essere affrontata in due modi:

- L'architettura database multi-tier può essere abbinata alla scheda ActiveX per distribuire un'applicazione client come controllo ActiveX. In questo modo, qualsiasi browser che supporta ActiveX è in grado di eseguire l'applicazione client come server in-process.
- Si possono utilizzare i pacchetti dati XML per generare un'applicazione InternetExpress. In questo modo si consente ai browser che supportano javascript di interagire con l'applicazione client tramite pagine html.

Queste due metodologie sono molto diverse. La scelta deve essere effettuata in base alle considerazioni seguenti:

- Entrambi i metodi utilizzano una diversa tecnologia (ActiveX rispetto a javascript e XML). Analizzare attentamente quali sistemi saranno utilizzati dagli utenti finali utilizzeranno. Il primo metodo richiede un browser che supporti ActiveX (il che limita i client all'uso dei una piattaforma Windows). Il secondo metodo richiede un browser che supporti javascript e le capacità DHTML introdotte da Netscape 4 e Internet Explorer 4.
- I controlli ActiveX devono essere scaricati nel browser per poter agire come server in-process. Da ciò ne consegue che i client che utilizzeranno il metodo ActiveX avranno bisogno di più memoria rispetto ai client di un'applicazione basata su html.
- Il metodo InternetExpress può essere integrato con altre pagine HTML. Un client ActiveX deve essere eseguito in una finestra separata.
- Il metodo InternetExpress utilizza lo standard HTTP, evitando in questo modo qualsiasi problema di firewall rispetto a un'applicazione ActiveX.

- Il metodo ActiveX offre una maggior flessibilità per ciò che concerne la programmazione dell'applicazione. Non si è limitati dalle capacità delle librerie javascript. I dataset client utilizzati con il metodo ActiveX mettono a disposizione funzionalità più estese (come filtri, intervalli, aggregazione, parametri facoltativi, recupero differito di BLOB, campi detail annidati, e così via) rispetto ai broker XML utilizzati con il metodo InternetExpress.



L'aspetto e il comportamento dell'applicazione client Web potrebbero essere differenti se l'applicazione viene utilizzata con browser differenti. È bene collaudare l'applicazione con quei browser che si prevede saranno utilizzati dagli utenti finali.

Distribuzione di un'applicazione client come controllo ActiveX

L'architettura database multi-tiered può essere abbinata a funzioni ActiveX per distribuire un'applicazione client sotto forma di controllo ActiveX.

Quando si distribuisce un'applicazione client come controllo ActiveX, viene creato l'application server come per qualsiasi altra applicazione multi-tier. Per informazioni dettagliate sulla creazione dell'application server, consultare "Creazione dell'application server" a pagina 29-12.

Quando si crea l'applicazione client, si deve usare come base una scheda *Active* Form al posto di una scheda ordinaria. Per informazioni dettagliate, consultare "Creazione di una Active Form per l'applicazione client".

Dopo aver costruito e distribuito l'applicazione client, è possibile accedervi da qualsiasi browser Web abilitato per ActiveX su un'altra macchina. Perché un browser Web lanci con successo l'applicazione client, il server Web deve essere in esecuzione sulla macchina che ha l'applicazione client.

Se l'applicazione client usa DCOM per le comunicazioni tra l'applicazione client e l'application server, la macchina con il browser Web deve essere in grado di lavorare con DCOM. Se la macchina con il browser Web è una macchina Windows 95, occorre avere installato DCOM95, che è disponibile tramite Microsoft.

Creazione di una Active Form per l'applicazione client

1. Poiché l'applicazione client verrà distribuita come controllo ActiveX, occorre avere un server Web che venga eseguito sullo stesso sistema dell'applicazione client. È possibile usare un server già pronto come Personal Web di Microsoft o scriverne uno proprio usando i componenti socket descritti nel Capitolo 37, "Operazioni con i socket".
2. Creare l'applicazione client seguendo le operazioni descritte nel paragrafo "Creazione dell'applicazione client" a pagina 29-23, iniziando, però, dalla selezione di File | New | ActiveX | Active Form, invece di iniziare un normale progetto client.
3. Se l'applicazione client usa un modulo dati, aggiungere una chiamata per crearlo esplicitamente nell'inizializzazione della scheda attiva.

4 Quando l'applicazione client ha terminato, compilare il progetto e selezionare Project|Web Deployment Options. Nella finestra di dialogo Web Deployment Options, occorre eseguire le seguenti operazioni:

   1 Nella pagina Project specificare la directory di destinazione, l'URL per la directory di destinazione e la directory HTML. Normalmente, la directory di destinazione e la directory HTML coincideranno con quella dei progetti per il server Web. L'URL di destinazione è solitamente il nome della macchina server.

   2 Nella pagina Additional Files includere midas.dll con l'applicazione client.

5 Infine, selezionare Project|WebDeploy per distribuire l'applicazione client come una scheda Active.

Qualsiasi browser Web che può eseguire schede *Active* può eseguire l'applicazione client specificando il file .HTM che è stato creato quando è stata distribuita l'applicazione client. Questo file .HTM ha lo stesso nome del progetto dell'applicazione client, e appare nella directory specificata come directory di destinazione.

Creazione di applicazioni Web con InternetExpress

Un'applicazione client può richiedere che l'application server fornisca pacchetti dati codificati in XML invece che come OleVariants. Combinando i pacchetti dati codificati in XML, speciali librerie javascript di funzioni database e il supporto per le applicazioni Web server, è possibile creare applicazioni thin client a cui si può accedere utilizzando un browser Web che supporta javascript. Questa combinazione di funzionalità è detta InternetExpress.

Prima di creare un'applicazione InternetExpress, è bene che sia chiara l'architettura delle applicazione Web server. Questa architettura è descritta nel Capitolo 32, "Creazione di applicazioni server per Internet".

Un'applicazione InternetExpress amplia l'architettura di base di un'applicazione per Web server in modo che agisca come il client di un application server. Le applicazioni InternetExpress generano pagine HTML che contengono un misto di HTML, XML e javascript. HTML pilota il layout e l'aspetto delle pagine viste dagli utenti finali nel loro browser. XML codifica i pacchetti dati e i pacchetti delta che rappresentano le informazioni del database. Javascript consente ai controlli HTML di interpretare e trattare i dati contenuti in questi pacchetti dati XML sulla macchina client.

Se l'applicazione InternetExpress utilizza DCOM per effettuare la connessone all'application server, si devono compiere altre operazioni per assicurarsi che l'application server garantisca ai propri client l'accesso e i permessi di esecuzione. Per maggiori informazioni, consultare "Concessione dei permessi per l'accesso e l'avvio dell'application server" a pagina 29-35.



È possibile creare applicazioni InternetExpress per fornire ai Web browsers dati "dal vivo"anche se non si possiede un application server. È sufficiente aggiungere al modulo web il provider e il dataset associato.

Creazione di un'applicazione InternetExpress

Le operazioni seguenti descrivono un modo per costruire un'applicazione Web utilizzando InternetExpress. Il risultato è un'applicazione che crea pagine HTML che consentono agli utenti di interagire con i dati di un application server mediante un browser Web che supporta javascript. È possibile anche costruire un'applicazione InternetExpress utilizzando l'architettura Site Express utilizzando il produttore di pagine di InternetExpress (*TInetXPageProducer*).

1. Scegliere File | New | Other per visualizzare la finestra di dialogo New Items, e sulla pagina New scegliere Web server application. Questo processo è descritto in "Creazione di applicazioni per Web server con Web Broker" a pagina 33-1.
2. Dalla pagina DataSnap della Component palette, aggiungere un componente connection al modulo web che appare quando si crea una nuova applicazione Web server. Il tipo di componente connessione che si aggiunge dipende dal protocollo di comunicazione che si vuole utilizzare. Per i dettagli, consultare la sezione "Scelta di un protocollo di connessione" a pagina 29-9.
3. Impostare le proprietà del componente connessione in modo da specificare l'application server con cui si dovrebbe istituire una connessione. Per ulteriori informazioni sulla configurazione dei componenti connessione, consultare la sezione "Connessione all'application server" a pagina 29-24.
4. Invece di un dataset client, dalla pagina InternetExpress della Component Palette aggiungere al modulo web un broker XML. Come *TClientDataSet*, *TXMLBroker* rappresenta i dati forniti da un provider sull'application server e interagisce con l'application server tramite la sua interfaccia *IAppServer*. Comunque, a differenza dei dataset client, i broker XML richiedono i pacchetti dati sotto forma di XML invece che come OleVariants, e interagiscono con componenti InternetExpress invece che con i controlli per dati.
5. Impostare la proprietà *RemoteServer* del broker XML in modo che punti al componente connection aggiunto al punto 2. Impostare la proprietà *ProviderName* in modo da indicare il provider sull'application server che fornisce i dati e applica gli aggiornamenti. Per maggiori informazioni sulla configurazione dei broker XML, consultare la sezione "Uso di un broker XML" a pagina 29-36.
6. Aggiungere al modulo web un produttore di pagine InternetExpress (*TInetXPageProducer*) per ogni singola pagina che gli utenti vedranno nel loro browser. Per ogni produttore di pagine, si dovrà impostare la proprietà *IncludePathURL* per indicare dove trovare le librerie javascript che introdurranno nei controlli HTML generati le funzionalità per la gestione dei dati.
7. Fare clic-destro su una pagina Web e scegliere il comando Action Editor per visualizzare l'Action editor. Aggiungere un elemento di azione per ogni messaggio che si desidera gestire dal browser. Associare i produttori di pagine aggiunti al passo 6 con queste azioni, impostandone la proprietà *Producer* o scrivendo del codice in un gestore di evento *OnAction*. Per maggiori informazioni sull'aggiunta di elementi di azione utilizzando l'Action editor, consultare la sezione "Aggiunta di azioni al dispatcher" a pagina 33-5.

8 Fare doppio clic su ogni pagina Web per visualizzare il Web page editor. (Si può attivare l'editor anche facendo clic nell'Object Inspector sul pulsante ellissi accanto alla proprietà *WebPageItems*). In questo editor è possibile aggiungere tutti gli elementi Web in modo, progettando così tutte le pagine che gli utenti vedranno nel proprio browser. Per maggiori informazioni sulla progettazione di pagine Web per l'applicazione InternetExpress, consultare la sezione "Creazione di pagine web con produttori di pagine InternetExpress" a pagina 29-39.

9 Generare l'applicazione Web. Una volta installata questa applicazione sul server web, i browser la potranno richiamare specificando, come parte ScriptName dell'URL, il nome dell'applicazione e, come parte Pathinfo, il nome del componente Web Page.

Uso delle librerie javascript

Le pagine HTML generate dai componenti InternetExpress e gli elementi Web che esse contengono fanno uso di varie librerie javascript fornite nella directory source/webmidas:

Tabella 29.3 Librerie javascript

Libreria	Descrizione
xmldom.js	Questa libreria è un parser XML compatibile con DOM scritto in javascript. Consente ai parser che non supportano XML di utilizzare i pacchetti dati XML. Si noti che non include il supporto per XML Islands, che sono supportati da IE5 e dalle versioni successive.
xmldb.js	Questa libreria definisce classi di accesso ai dati che gestisce i pacchetti dati XML e i pacchetti delta XML.
xmldisp.js	Questa libreria definisce classi che associano le classi di accesso ai dati in xmldb con i controlli HTML nella pagina HTML.
xmlerrdisp.js	Questa libreria definisce classi che possono essere utilizzate durante la riconciliazione degli errori di aggiornamento. Queste classi non sono utilizzate da nessuno dei componenti interni InternetExpress, ma risultano utili quando si scriverà un produttore di riconciliazione.
xmlshow.js	Questa libreria include funzioni per visualizzare pacchetti dati XML e pacchetti delta XML formattati. Questa libreria non è utilizzata da nessun componente InternetExpress, ma è utile in fase di debug.

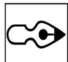
Una volta installate queste librerie, si deve impostare la proprietà *IncludePathURL* di tutti i produttori di pagine InternetExpress per indicare dove è possibile reperirle.

È possibile scrivere le proprie pagine HTML utilizzando le classi javascript fornite in queste librerie invece di utilizzare elementi Web per generare le pagine Web. Comunque, occorre essere certi che il codice scritto non faccia niente di illegale, in quanto queste classi includono un ridotto controllo sugli errori (per ridurre al minimo la dimensione delle pagine Web generate).

Concessione dei permessi per l'accesso e l'avvio dell'application server

Le richieste provenienti dalle applicazioni InternetExpress appaiono all'application server come se fossero originate da un account guest di nome IUSR_computername, dove computername è il nome del sistema che esegue l'applicazione Web. Per

impostazione predefinita, questo account non gode dei per messi per accedere o avviare l'application server. Se si cerca di utilizzare l'applicazione Web senza concedere questi permessi, nel momento in cui il browser Web tenta di caricare la pagina viene generato un errore `EOLE_ACCESS_ERROR`.



Poiché l'application server viene eseguito in questo account guest, non può essere disattivato dagli altri account.

Per accordare all'applicazione Web i permessi di accesso e di avvio, eseguire DCOMCnfg.exe, ubicato nella directory System32 della macchina che esegue l'application server. Le istruzioni seguenti descrivono come configurare l'application server:

1. Quando si esegue DCOMCnfg, selezionare l'application server nell'elenco delle applicazioni sulla pagina Applications.
2. Fare clic sul pulsante Properties. Non appena la finestra di dialogo cambia, selezionare la pagina Security.
3. Selezionare Use Custom Access Permissions, e premere il pulsante Edit. Aggiungere il nome IUSR_computername all'elenco degli account con permesso di accesso, dove computername è il nome della macchina che esegue l'applicazione Web.
4. Selezionare Use Custom Launch Permissions, e premere il pulsante Edit. Aggiungere IUSR_computername anche a questo elenco.
5. Fare clic sul pulsante Apply.

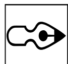
Uso di un broker XML

Il broker XML assolve due funzioni principali:

- Recupera i pacchetti dati XML dall'application server e li rende disponibili agli elementi Web che generano il codice HTML dell'applicazione InternetExpress.
- Riceve dai browser gli aggiornamenti sotto forma di pacchetti delta XML e li applica all'application server.

Recupero di pacchetti dati XML

Prima che il broker XML possa fornire i pacchetti dati XML ai componenti che generano le pagine HTML, deve recuperarli dall'application server. A questo scopo utilizza l'interfaccia *IAppServer* che acquisisce tramite un componente connessione.



Anche quando si utilizza SOAP, in cui l'application server supporta *IAppServerSOAP*, il broker XML utilizza *IAppServer* perché il componente connection agisce come un adattatore tra le due interfacce.

Per fare in modo che il produttore XML possa utilizzare l'interfaccia *IAppServer*, si devono impostare le seguenti proprietà:

- Impostare la proprietà *RemoteServer* al componente connessione che istituisce la connessione con l'application server e ottiene la sua interfaccia *IAppServer*. In fase

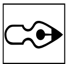
di progettazione, si può selezionare questo valore da un elenco a discesa nell'Object Inspector.

- Impostare la proprietà *ProviderName* al nome del componente provider sull'application server che rappresenta il dataset da cui si vogliono i pacchetti dati XML. Questo provider provvede a fornire i pacchetti dati XML e ad applicare gli aggiornamenti dai pacchetti delta XML. In fase di progettazione, se la proprietà *RemoteServer* è impostata e il componente connessione ha una connessione attiva, l'Object Inspector visualizza un elenco di provider disponibili. (Se si sta utilizzando una connessione DCOM l'application server deve essere registrato anche sulla macchina client).

Due proprietà consentono di indicare ciò che si desidera includere nei pacchetti dati:

- È possibile limitare il numero di record che vengono aggiunti al pacchetto dati impostando la proprietà *MaxRecords*. Ciò risulta particolarmente importante per important dataset estesi in quanto le applicazioni InternetExpress inviano l'intero pacchetto dati ai browser client. Se il pacchetto dati è troppo grande, il tempo necessario per scaricare i dati potrebbe diventare proibitivamente lungo.
- Se il provider sull'application server rappresenta una query o una procedura registrata, si potrebbero fornire valori di parametro prima di ottenere un pacchetto dati XML. Questi valori di parametro possono essere forniti mediante la proprietà *Params*.

I componenti che generano HTML e javascript per l'applicazione di InternetExpress, una volta impostata la loro proprietà *XMLBroker*, usano automaticamente il pacchetto dati XML del broker XML. Per ottenere direttamente il pacchetto dati XML nel codice, utilizzare il metodo *RequestRecords*.



Quando il broker XML fornisce un pacchetto dati a un altro componente (o si chiama il metodo *RequestRecords*), il broker riceve un evento *OnRequestRecords*. Si può utilizzare questo evento per fornire una propria stringa XML invece del pacchetto dati ottenuto dall'application server. Per esempio, si potrebbe recuperare il pacchetto dati XML dall'application server utilizzando *GetXMLRecords*, e quindi modificare il pacchetto prima di passarlo alla successiva pagina Web.

Applicazione degli aggiornamenti da pacchetti delta XML

Quando si aggiunge il broker XML al modulo web (o a un modulo dati contenente un *TWebDispatcher*), esso si registra automaticamente presso il dispatcher Web come un oggetto con dispatch automatico. Ciò significa che, a differenza di altri componenti, non è necessario creare un elemento di azione per il broker XML per far sì che sia in grado di rispondere ai messaggi di aggiornamento provenienti da un browser Web. Questi messaggi contengono pacchetti delta XML che dovrebbero essere applicati all'application server. Tipicamente, sono generati da un pulsante appositamente creato in una delle pagine HTML prodotte dall'applicazione client Web.

Affinché il dispatcher sia in grado di riconoscere i messaggi per il broker XML, è necessario descriverli utilizzando la proprietà *WebDispatch*. Impostare la proprietà *PathInfo* alla sezione "percorso" dell'URL a cui vengono trasmessi i messaggi per il broker XML. Impostare *MethodType* al valore dell'header del metodo dei messaggi di aggiornamento indirizzati a quell'URL (di solito *mtPost*). Se si vuole rispondere a

tutti i messaggi aventi il percorso specificato, impostare *MethodType* a *mtAny*. Se non si vuole che il broker XML risponda direttamente ai messaggi di aggiornamento (per esempio, nel caso li si voglia gestire esplicitamente utilizzando un elemento di azione), impostare la proprietà *Enabled* a **false**. Per maggiori informazioni su come fa il dispatcher Web a determinare il componente che gestirà i messaggi provenienti dal browser Web, consultare la sezione "Dispatch dei messaggi di richiesta" a pagina 33-5.

Quando il dispatcher passa un messaggio di aggiornamento al broker XML, passa gli aggiornamenti all'application server e, se si verificano errori di aggiornamento, riceve un pacchetto delta XML che descrive tutti gli errori di aggiornamento. Infine, ritrasmette al browser un messaggio di risposta che può sia indirizzare di nuovo il browser alla stessa pagina che aveva generato il pacchetto delta XML, sia trasmettergli nuovi contenuti.

Esistono diversi eventi che consentono di inserire elaborazione personalizzate in tutte queste fasi del processo di aggiornamento:

1 Quando il dispatcher passa per la prima volta il messaggio di aggiornamento al broker XML, riceve un evento *BeforeDispatch*, in cui è possibile preelaborare la richiesta o anche gestirla completamente. Questo evento consente al broker XML di gestire messaggi diversi da quelli di aggiornamento.

2 Se il gestore di evento *BeforeDispatch* non gestisce il messaggio, il broker XML riceve un evento *OnRequestUpdate*, in cui è possibile applicare gli aggiornamenti personalmente invece di utilizzare l'elaborazione predefinita.

3 Se il gestore di evento *OnRequestUpdate* non gestisce la richiesta, il broker XML applica gli aggiornamenti e riceve un pacchetto delta che contiene tutti gli errori di aggiornamento.

4 Se non si verificano errori di aggiornamento, il broker XML riceve un evento *OnGetResponse*, in cui è possibile creare un messaggio di risposta che indica che gli aggiornamenti sono stati applicati con successo oppure trasmettere al browser i dati aggiornati. Se il gestore di evento *OnGetResponse* non completa la risposta (non imposta il parametro *Handled* a true), il broker XML trasmette una risposta che dirige di nuovo il browser al documento che aveva generato il pacchetto delta.

5 Se si verificano errori di aggiornamento, il broker XML riceve invece un evento *OnGetErrorResponse*. È possibile utilizzare questo evento per tentare di risolvere gli errori di aggiornamento oppure per generare una pagina Web che li descriva all'utente. Se il gestore di evento *OnGetErrorResponse* non completa la risposta (non imposta il parametro di dled a true), il broker XML chiama uno speciale produttore di contenuti, di nome *ReconcileProducer*, per generare il contenuto del messaggio di risposta.

6 Infine, il broker XML riceve un evento *AfterDispatch*, in cui è possibile compiere eventuali operazioni finali prima di ritrasmettere una risposta al browser Web.

Creazione di pagine web con produttori di pagine InternetExpress

Ogni produttore di pagine InternetExpress genera un documento HTML che appare nel browser dei client dell'applicazione. Se l'applicazione include diversi documenti Web distinti, utilizzare per ognuna di esse un diverso produttore di pagine.

Il produttore di pagina InternetExpress (*TInetXPageProducer*) è uno speciale componente produttore di pagine. Come gli altri produttori di pagine, è possibile assegnarlo alla proprietà *Producer* di un elemento di azione, oppure lo può chiamare esplicitamente da un gestore di evento *OnAction*. Per maggiori informazioni sull'utilizzo di produttori di contenuti con elementi di azione, consultare la sezione "Risposta a messaggi di richiesta mediante elementi di azione" a pagina 33-8. Per maggiori informazioni sui produttori di pagine, consultare la sezione "Utilizzo di componenti produttori di pagine" a pagina 33-14.

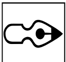
Il produttore di pagine InternetExpress ha, come valore della proprietà di *HTMLDoc*, un modello predefinito. Questo modello contiene una serie di marcatori, trasparenti per HTML, che il produttore di pagine InternetExpress utilizza per assemblare un documento HTML (che incorpora javascript e XML) includendo i contenuti prodotti da altri componenti. Prima che esso possa tradurre tutti i marcatori trasparenti per HTML e assemblare questo documento, è necessario indicare l'ubicazione delle librerie javascript utilizzate per il codice javascript incorporato nella pagina. Questa posizione viene specificata impostando la proprietà *IncludePathURL*.

È possibile specificare i componenti che generano parti della pagina Web utilizzando il Web page editor. Visualizzare il Web page editor facendo doppio clic sul componente Web page oppure facendo clic sul pulsante ellissi accanto alla proprietà *WebPageItems* nell'Object Inspector.

I componenti aggiunti nel Web page editor generano il codice HTML che sostituisce uno dei marcatori trasparenti per HTML inclusi nel modello predefinito del produttore di pagine InternetExpress. Questi componenti diventano il valore della proprietà *WebPageItems*. Dopo avere aggiunto i componenti nell'ordine voluto, è possibile personalizzare il modello aggiungendo del codice HTML o modificando i marcatori predefiniti.

Utilizzo di Web page editor

Web page editor permette di aggiungere elementi Web al produttore di pagine InternetExpress e di vedere la pagina HTML risultante. Visualizzare il Web page editor facendo doppio clic su un componente produttore di pagine InternetExpress.



Per utilizzare Web page editor, è necessario installare Internet Explorer 4 o una versione successiva.

La zona superiore del Web page editor visualizza gli elementi Web che generano il documento HTML. Questi elementi Web sono annidati, e in particolare ogni tipo di elemento Web assembla il codice HTML generato dai propri sottoelementi. Tipi diversi di elementi possono contenere sottoelementi differenti. Sulla sinistra, una struttura ad albero visualizza tutti degli elementi Web, indicando come sono annidati. Sulla destra, si possono vedere gli elementi Web inclusi dall'elemento al

momento selezionato. Se si seleziona un componente nella parte superiore del Web page editor, è possibile impostarne le proprietà utilizzando l'Object Inspector.

Fare clic sul pulsante New Item per aggiungere un sottoelemento all'elemento attualmente selezionato. La finestra di dialogo Add Web Component elenca solo quegli elementi che possono essere aggiunti all'elemento attualmente selezionato.

Il produttore di pagine InternetExpress può contenere un solo tipo di elemento, dei due possibili, ognuno dei quali genera una scheda HTML:

- *TDataForm*, che genera una scheda HTML per visualizzare dati e i controlli che trattano tali dati o che trasmettono gli aggiornamenti.

  Gli elementi che si aggiungono a *TDataForm* visualizzano i dati in una griglia multi-record (*TDataGrid*) o in una serie di controlli, ognuno dei quali rappresenta un singolo campo di un singolo record (*TFieldGroup*). Inoltre, è possibile aggiungere una serie di pulsanti per spostarsi tra i dati o per trasmette gli aggiornamenti, (*TDataNavigator*), o un pulsante per applicare gli aggiornamenti al client Web (*TApplyUpdatesButton*). Ognuno di questi elementi contiene dei sottoelementi per rappresentare singoli campi o pulsanti. Infine, come con la maggior parte degli elementi Web, è possibile aggiungere una griglia di layout, (*TLayoutGroup*), che permette di personalizzare il layout di tutti gli elementi in essa contenuti.

- *TQueryForm*, che genera una scheda HTML per visualizzare o leggere i valori definiti dall'applicazione. Per esempio, si può utilizzare questa scheda per visualizzare e trasmettere i valori dei parametri.

  Gli elementi che si aggiungono a *TQueryForm* visualizzano i valori definiti dall'applicazione (*TQueryFieldGroup*) o una serie di pulsanti per trasmettere o azzerare quei valori (*TQueryButtons*). Ognuno di questi elementi contiene dei sottoelementi per rappresentare singoli valori o pulsanti. Anche ad una scheda di interrogazione, è possibile aggiungere una griglia di layout, esattamente come per le schede per dati.

La parte inferiore del Web page editor visualizza il codice HTML generato e permette di vederne l'aspetto finale come in un browser (IE4).

Impostazione delle proprietà degli elementi Web

Gli elementi Web che si aggiungono utilizzando il Web page editor sono componenti specializzati che generano HTML. Ogni classe di elemento Web è stata progettata di produrre un determinato controllo o sezione del documento HTML finale, ma una serie comune di proprietà influenza l'aspetto dell'HTML finale.

Quando un elemento Web rappresenta le informazioni di un pacchetto dati XML (per esempio, quando genera una serie di campi o di controlli per la visualizzazione dei parametri o un pulsante che interviene sui dati), la proprietà *XMLBroker* associa l'elemento Web col broker XML che gestisce il pacchetto dati. Usando la proprietà *XMLDataSetField* si può anche specificare un dataset, che è contenuto in un campo del dataset di quel pacchetto dati. Se l'elemento Web rappresenta un certo campo o un valore di parametro, l'elemento Web ha una proprietà *FieldName* o *ParamName*.

È possibile applicare un attributo di stile a qualsiasi elemento Web, influenzando in questo modo l'aspetto complessivo di tutto l' HTML generato. Stili e fogli di stile fanno parte degli standard di HTML 4. Essi consentono a un documento HTML di definire una serie di attributi di visualizzazione che vengono applicati a un marcatore e a tutto ciò che è incluso nel suo campo d'azione. Gli elementi Web ne consentono un uso molto flessibile:

- Il modo più semplice per utilizzare gli stili è quello di definire direttamente un attributo di stile sull'elemento Web. A questo scopo, si utilizza la proprietà *Style*. Il valore di *Style* è semplicemente la porzione di definizione dell'attributo di una definizione standard di stile HTML, come, ad esempio
  `color: red.`
- È possibile anche definire un foglio di stile, che definisce una serie di definizioni di stile. Ogni definizione include sia un selettore di stile (il nome di un marcatore a cui viene sempre applicato lo stile oppure un nome di stile definito dall'utente) sia la definizione dell'attributo racchiusa tra parentesi graffe:
  ```
  H2 B  {color: red}
  .MyStyle  {font-family: arial; font-weight: bold; font-size: 18px }
  ```
  L'intero set di definizioni è conservata dal produttore di pagine InternetExpress nella sua proprietà *Styles*. Ogni elemento Web può referenziare quindi gli stili con nomi definiti dall'utente impostando la sua proprietà *StyleRule*.
- Se si sta condividendo un foglio di stile con altre applicazioni, è possibile fornire le definizioni di stile come valore della proprietà *StylesFile* del produttore di pagine InternetExpress invece che della proprietà *Style*. I singoli elementi Web faranno riferimento agli stili utilizzando sempre la proprietà *StyleRule*.

Un'altra proprietà comune degli elementi Web è la proprietà *Custom*. *Custom* include una serie di opzioni che è possibile aggiungere al marcatore HTML generato. HTML definisce una diversa serie di opzioni per ogni tipo di marcatore. Il manuale Visual Component Library Reference riporta per la proprietà *Custom* di vari elementi Web un esempio delle possibili opzioni. Per maggiori informazioni sulle possibili opzioni, utilizzare un manuale di riferimento alla sintassi HTML.

Personalizzazione del modello del produttore di pagine InternetExpress

Il modello di un produttore di pagine InternetExpress è un documento HTML che incorpora marcatori aggiuntivi, tradotti dinamicamente dall'applicazione. Inizialmente, il produttore di pagine genera un modello predefinito come valore della proprietà *HTMLDoc*. Questo modello predefinito ha il seguente aspetto

```
<HTML>
<HEAD>
</HEAD>
<BODY>
<#INCLUDES> <#STYLES> <#WARNINGS> <#FORMS> <#SCRIPT>
</BODY>
</HTML>
```

I marcatori trasparenti per HTML inclusi nel modello predefinito vengono tradotti nel seguente modo:

<#INCLUDES> genera le istruzioni che includono le librerie javascript. Queste istruzioni assumono il seguente aspetto

```
<SCRIPT language=Javascript type="text/javascript" SRC="IncludePathURL/xmldom.js"> </SCRIPT>
<SCRIPT language=Javascript type="text/javascript" SRC="IncludePathURL/xmldb.js"> </SCRIPT>
<SCRIPT language=Javascript type="text/javascript" SRC="IncludePathURL/xmlbind.js"> </SCRIPT>
```

<#STYLES> genera le istruzioni che definiscono un foglio di stile a partire dalle definizioni elencate nella proprietà *Style* o nella proprietà *StylesFile* del produttore di pagine InternetExpress.

<#WARNINGS> non genera nulla in fase di esecuzione. In fase di progettazione, aggiunge messaggi di avvertimento per problemi rilevati durante la generazione del documento HTML. È possibile visualizzare questi messaggi nel Web page editor.

<#FORMS> genera il codice HTML prodotto dai componenti aggiunti nel Web page editor. Il codice HTML prodotto da ogni componente è generato in base alla posizione che occupa nella sequenza di *WebPageItems*.

<#SCRIPT> genera un blocco di dichiarazioni javascript che sono utilizzate nel codice HTML generato dai componenti aggiunti nel Web page editor.

È possibile sostituire il modello predefinito modificando il valore di *HTMLDoc* oppure impostando la proprietà *HTMLFile*. Il modello HTML personalizzato può includere tutti i marcatori trasparenti per HTML che costituiscono il modello predefinito. Il produttore di pagine InternetExpress tradurrà automaticamente questi marcatori nel momento in cui si chiama il metodo *Content*. Inoltre, il produttore di pagine InternetExpress è in grado di tradurre automaticamente altri tre marcatori:

<#BODYELEMENTS> viene sostituito dallo stesso codice HTML prodotto dai 5 marcatori inclusi nel modello predefinito. Risulta utile quando si genera un modello in un editor HTML e si desidera utilizzare il layout predefinito aggiungendo però altri elementi con l'editor.

<#COMPONENT Name=WebComponentName> viene sostituito dal codice HTML generato dal componente con nome *WebComponentName*. Questo componente può essere uno dei componenti aggiunti nel Web page editor, o un qualsiasi altro componente purché supporti l'interfaccia *IWebContent* e abbia lo stesso Owner del produttore di pagine InternetExpress.

<#DATAPACKET XMLBroker=BrokerName> viene sostituito col pacchetto dati XML ottenuto dal broker XML specificato con *BrokerName*. Quando si osserva nel Web page editor il codice HTML generato dal produttore di pagine InternetExpress, si vedrà questo marcatore invece dell'effettivo pacchetto dati XML.

Inoltre, il modello personalizzato può includere qualsiasi altro marcatore trasparente per HTML definito dall'utente. Quando il produttore di pagine InternetExpress incontra un marcatore che non appartiene a nessuno dei sette tipi che è in grado di tradurre automaticamente, genera un evento *OnHTMLTag* in cui si potrà inserire del codice che esegua traduzioni ad hoc. Per maggiori informazioni sui modelli HTML in generale, consultare la sezione "Modelli HTML" a pagina 33-14.



I componenti che appaiono nel Web page editor generano codice statico. ciò significa che, a meno che l'application server modifichi i metadati inclusi nei pacchetti dati, il codice HTML è sempre lo stesso, indipendentemente da quando viene generato. È possibile evitare il sovraccarico di lavoro derivante dalla generazione dinamica in fase di esecuzione di questo codice, in risposta ad ogni messaggio di richiesta, se si copia il codice HTML generato nel Web page editor e lo si utilizza come modello. Poiché il Web page editor visualizza un marcatore <#DATAPACKET> invece dell'effettivo XML, l'utilizzo di questo codice come modello continua a consentire all'applicazione di recuperare dinamicamente i pacchetti dati dall'application server.

Capitolo

30

Uso di XML nelle applicazioni database

Oltre al supporto per la connessione ai server di database, C++Builder permette di gestire documenti XML come se anch'essi fossero server di database. XML (Extensible Markup Language) è un linguaggio di markup per la descrizione di dati strutturati. I documenti XML dotano di un formato standard e trasportabile i dati utilizzati nelle applicazioni Web, nelle comunicazioni business-to-business, e così via. Per informazioni sulle possibilità offerte da C++Builder per lavorare direttamente con i documenti XML, consultare il Capitolo 35, "Operazioni con documenti XML".

Le possibilità offerte da C++Builder per operare con documenti XML nelle applicazioni database si basano su un insieme di componenti in grado di convertire dei pacchetti dati (la proprietà *Data* di un client dataset) in documenti XML e, inversamente, di convertire documenti XML in pacchetti dati. Prima di utilizzare questi componenti, bisogna definire la trasformazione tra il documento XML e il pacchetto dati. Una volta definita la trasformazione, i componenti speciali si possono usare per

- convertire documenti XML in pacchetti dati.
- fornire dati prelevati da un documento XML e risolvere gli aggiornamenti a un documento XML.
- utilizzare un documento XML come client di un provider.

Definizione della trasformazione

Prima di convertire tra pacchetti dati e documenti XML, bisogna definire la relazione tra i metadati in un pacchetto dati e i nodi del documento XML corrispondente. La descrizione della relazione è registrata in uno speciale documento XML detto trasformazione.

Ogni file di trasformazione contiene due cose: la mappatura tra i nodi di uno schema XML e i campi di un pacchetto dati, più uno schema di documento XML che rappresenta la struttura dei risultati della trasformazione. Una trasformazione è una mappatura a senso unico: da uno schema o da un documento XML a un pacchetto dati, oppure dai metadati di un pacchetto dati a uno schema XML. Spesso, si creano file di trasformazione a coppie: uno che fa una mappatura da XML a pacchetto dati e un altro che mappa da pacchetto dati a XML.

Per creare i file di trasformazione di una mappatura, si ricorre al programma di utilità XMLMapper fornito nella directory bin.

Mappatura tra nodi XML e campi di un pacchetto dati

XML dispone di una modalità in formato testo per memorizzare o rappresentare i dati strutturati. I dataset forniscono un'altra modalità per memorizzare e descrivere i dati strutturati. Per convertire un documento XML in un dataset, quindi, è necessario identificare le corrispondenze tra i nodi di un documento XML e i campi di un dataset.

Si consideri, ad esempio, un documento XML che rappresenta un insieme di messaggi di posta elettronica. Potrebbe presentarsi così (con un solo messaggio):

```
<?xml version="1.0" standalone='yes' ?>
<email>
   <head>
      <from>
         <name>Dave Boss</name>
         <address>dboss@MyCo.com</address>
      </from>
      <to>
         <name>Joe Engineer</name>
         <address>jengineer@MyCo.com</address>
      </to>
      <cc>
         <name>Robin Smith/name>
         <address>rsmith@MyCo.com</address>
      </cc>
      <cc>
         <name>Leonard Devon</name>
         <address>ldevon@MyCo.com</address>
      </cc>
   </head>
   <body>
      <subject>XML components</subject>
      <content>
        Joe,
        Attached is the specification for the new XML component support in C++Builder.
        This looks like a good solution to our buisness-to-buisness application!
        Also attached, please find the project schedule. Do you think its reasonable?
           Dave.
      </content>
      <attachment attachfile="XMLSpec.txt"/>
      <attachment attachfile="Schedule.txt"/>
```

```
    </body>
  </email>
```

Un modo naturale di mappatura tra il documento e un dataset fa corrispondere un unico record a ciascun messaggio di posta elettronica. Il record conterrà i campi del nome e dell'indirizzo del mittente. Siccome i destinatari del messaggio potrebbero essere più di uno, il destinatario (<to>) dovrebbe essere mappato a un dataset annidato. La riga <subject> potrebbe essere mappata a un campo stringa, mentre probabilmente il messaggio stesso (<content>) potrebbe essere un campo memo. I nomi dei file allegati potrebbero essere mappati a un dataset annidato perché un messaggio può avere numerosi allegati. < Pertanto, il messaggio precedente potrebbe essere mappato a un dataset simile al seguente:

SenderName	SenderAddress	To	CC	Subject	Content	Attach
Dave Boss	dboss@MyCo.Com	(DataSet)	(DataSet)	XML components	(MEMO)	(DataSet)

in cui il dataset annidato nel campo "To" è

Nome	Indirizzo
Joe Engineer	jengineer@MyCo.Com

il dataset annidato nel campo "CC" è

Nome	Indirizzo
Robin Smith	rsmith@MyCo.Com
Leonard Devon	ldevon@MyCo.Com

e il dataset annidato nel campo "Attach" è

Attachfile
XMLSpec.txt
Schedule.txt

Definire questa mappatura significa identificare i nodi del documento XML che possono essere ripetuti e farli corrispondere con dataset annidati. Gli elementi provvisti di tag e che hanno un valore vengono visualizzati solo una volta (come <content>...</content>) corrispondono a campi il cui tipo di dato riflette il tipo di dati che può venire visualizzato come valore. Gli attributi di un tag (come l'attributo AttachFile del tag degli allegati) sono in grado di mapparsi anche con dei campi.

Va notato che non tutti i tag nel documento XML appaiono nel dataset corrispondente. Ad esempio, all'elemento <head>...<head/> non corrisponde alcun elemento nel dataset risultante. Di solito, solo elementi che hanno valori, elementi che possono essere ripetuti o gli attributi di un tag corrispondono a campi (compresi i campi di dataset annidati) di un dataset. L'eccezione a questa regola si ha quando un nodo genitore del documento XML corrisponde a un campo il cui valore è dato dai valori dei nodi figlio. Ad esempio, un documento XML potrebbe contenere un insieme di tag come

```
<FullName>
   <Title> Mr. </Title>
   <FirstName> John </FirstName>
   <LastName> Smith </LastName>
</FullName>
```

che potrebbe corrispondere a un singolo campo dataset con il valore

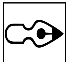
```
Mr. John Smith
```

Utilizzo di XMLMapper

Il programma di utilità di mappatura XML, xmlmapper.exe, permette di definire le mappature in tre modi:

- Da uno schema XML (o documento) esistente a un dataset client che si definisce. Serve per creare un'applicazione di database che funzioni con dati di cui già si ha uno schema XML.
- Da un pacchetto dati esistente a un nuovo schema XML da definire. È utile quando si desidera esporre informazioni di database esistenti in XML, ad esempio per creare un nuovo sistema di comunicazione business-to-business.
- Tra uno schema XML esistente e un pacchetto dati esistente. È utile quando si desidera fare funzionare insieme uno schema XML e un database che descrivono le stesse informazioni.

Una volta definita la mappatura, è possibile generare i file di trasformazione che servono a convertire i documenti XML in pacchetti dati e, inversamente, a convertire i pacchetti dati in documenti XML. Notare che solo il file di trasformazione è direzionale: una singola mappatura serve per generare sia la trasformazione da XML a pacchetto dati, sia quella da pacchetto dati a XML.



XML mapper si appoggia a due file .DLL (midas.dll e msxml.dll) per funzionare correttamente. Accertarsi che ambedue i file siano stati installati prima di usare xmlmapper.exe. Inoltre, msxml.dll deve essere registrato come COM server. Per registrarlo si utilizza Regsvr32.exe.

Caricamento di uno schema XML o di un pacchetto dati

Prima di definire una mappatura e di generare un file di trasformazione, bisogna caricare le descrizioni del documento XML e del pacchetto dati di cui si sta tracciando la mappa.

È possibile caricare un documento o uno schema XML selezionando File|Open e poi il documento o lo schema nella finestra di dialogo che appare.

È possibile caricare un pacchetto dati scegliendo File|Open e selezionando un file di pacchetto dati nella finestra di dialogo che appare. (il pacchetto dati è semplicemente il file generato quando si chiama il metodo *SaveToFile* di un dataset client). Se non si è salvato il pacchetto dati su disco, lo si può prelevare direttamente dall'application server di un'applicazione multi-tier facendo clic con il pulsante destro nel riquadro Datapacket e scegliendo Connect To Remote Server.

È possibile caricare solo un documento o uno schema XML, solo un pacchetto dati, oppure caricarli entrambi. Se si carica solo un lato della mappatura, XML mapper può generare una mappatura naturale dell'altro lato.

Definizione delle mappature

La mappatura tra un documento XML e un pacchetto dati non deve obbligatoriamente comprendere tutti i campi del pacchetto dati o tutti gli elementi dotati di tag del documento XML. Quindi, bisogna prima specificare quali elementi mappare. Per specificarli, selezionare prima la pagina Mapping nel riquadro centrale della finestra di dialogo.

Per specificare gli elementi di un documento o di uno schema XML mappati nei campi di un pacchetto dati, si seleziona la pagina DocumentView o SchemaView del riquadro del documento XML e si fa doppio clic sui nodi degli elementi che mappano nei campi del pacchetto dati.

Specificare i campi del pacchetto dati mappati verso gli elementi o attributi etichettati nel documento XML e fare doppio clic sui nodi relativi ai campi nel riquadro Datapacket.

Se si è solo caricato un lato della mappatura (il documento XML o il pacchetto dati), è possibile generare l'altro dopo avere selezionato i nodi mappati.

- Se si genera un pacchetto dati da un documento XML, prima si definiscono gli attributi dei nodi selezionati, che determinano i tipi di campi a cui essi corrispondono nel pacchetto dati. Nel riquadro centrale, si seleziona la pagina Node Repository. Selezionare ogni nodo che partecipa alla mappatura e indicare gli attributi del campo corrispondente. Se la mappatura non è diretta (ad esempio, un nodo con sottonodi che corrisponde a un campo il cui valore è determinato dai sottonodi), spuntare la casella di controllo User Defined Translation. Sarà necessario poi scrivere un gestore di evento per eseguire la trasformazione sui nodi definiti dall'utente.

  Una volta che si è specificato come devono essere mappati i nodi, scegliere Create| Datapacket da XML. Il pacchetto dati corrispondente viene automaticamente generato e appare nel riquadro Datapacket.

- Se si genera un documento XML da un pacchetto dati, scegliere Create|XML da Datapacket. Viene visualizzata una finestra di dialogo dove è possibile specificare i nomi dei tag e degli attributi nel documento XML che corrispondono a campi, a record e a dataset nel pacchetto dati. Per valori del campo, si specifica se, dal nome che gli si dà, corrispondono a un elemento con tag, dotato di un valore, o a un attributo. Nomi che iniziano con il simbolo @ mappano ad attributi del tag corrispondenti al record, mentre nomi che non iniziano Il simbolo @ mappano a elementi dotati di tag che hanno valori e che sono annidati all'interno dell'elemento del record.
- Se si sono caricati sia un documento XML che un pacchetto dati (file di dataset client), assicurarsi di selezionare i nodi corrispondenti nello stesso ordine. I nodi corrispondenti dovrebbero apparire l'uno accanto all'altro nella tabella all'inizio della pagina Mapping.

Una volta che si è caricato o generato sia il documento XML che il pacchetto dati e che si sono selezionati i nodi visualizzati nella mappatura, la tabella all'inizio della pagina Mapping riflette la mappatura che si è definita.

Generazione dei file di trasformazione

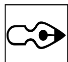
Per generare un file di trasformazione, utilizzare i seguenti passi:

1. Per prima cosa, selezionare il pulsante di scelta indicante che cosa si ottiene dalla trasformazione:
    - Scegliere il pulsante Datapacket to XML se la mappatura va da pacchetto dati a documento XML.
    - Scegliere il pulsante XML to Datapacket se la mappatura va da documento XML a pacchetto dati.
2. Se si genera un pacchetto dati, si desidererà anche utilizzare i pulsanti di scelta nella sezione Create Datapacket As. Questi pulsanti permettono di specificare come sarà utilizzato il pacchetto dati: come dataset, come pacchetto delta per applicare aggiornamenti, come parametri da fornire a un provider prima di fornire i dati.
3. Fare clic su Create e su Test Transformation per generare una versione in memoria della trasformazione. XML mapper mostra il documento XML che verrebbe generato nel riquadro Datapacket o il pacchetto dati del documento XML nel riquadro XML Document.
4. Infine, scegliere File|Save|Transformation e salvare il file di trasformazione. I file di trasformazione sono file XML speciali (con l'estensione .xtr) che descrivono la trasformazione definita.

Conversione di documenti XML in pacchetti dati

Una volta creato un file di trasformazione indicante come trasformare un documento XML in un pacchetto dati, si possono creare pacchetti dati per qualsiasi documento XML conforme allo schema utilizzato nella trasformazione. Questi pacchetti dati possono quindi venire assegnati a un dataset client e salvati su un file, in modo da formare la base di un'applicazione per database basata su file.

Il componente *TXMLTransform* trasforma un documento XML in un pacchetto dati secondo la mappatura contenuta in un file di trasformazione.



È possibile anche utilizzare *TXMLTransform* per convertire un pacchetto dati visualizzato in formato XML in un documento XML arbitrario.

Specifica del documento XML di origine

Ci sono tre modi per specificare il documento XML di origine:

- Se il documento di origine è un file .xml su disco, è possibile utilizzare la proprietà *SourceXmlFile*.

- Se il documento di origine è una stringa in memoria di dati XML, è possibile utilizzare la proprietà *SourceXml*.
- Se per il documento di origine ha un'interfaccia *IDOMDocument*, è possibile utilizzare la proprietà *SourceXmlDocument*.

TXMLTransform controlla queste proprietà nell'ordine elencato. Solo se *SourceXmlFile* è una stringa vuota controlla la proprietà *SourceXml*. Solo se *SourceXmlFile* è una stringa vuota controlla la proprietà *SourceXml*. E solo se *SourceXml* è una stringa vuota controlla la proprietà *SourceXmlDocument*.

Specifica della trasformazione

Ci sono due modi per specificare la trasformazione che converte il documento XML in un pacchetto dati:

- Impostare la proprietà *TransformationFile* per indicare un file di trasformazione che è stato creato utilizzando xmlmapper.exe.
- Impostare la proprietà *TransformationDocument* se l'interfaccia della trasformazione è *IDOMDocument*.

TXMLTransform controlla queste proprietà nell'ordine elencato. Cioè, prima controlla il nome file nella proprietà *TransformationFile*. Solo se *TransformationFile* è una stringa vuota controlla la proprietà *TransformationDocument*.

Ottenimento del pacchetto dati risultante

Per fare in modo che *TXMLTransform* esegua la sua trasformazione e generi un pacchetto dati, è solamente necessario leggere la proprietà *Data*. Ad esempio, il seguente codice utilizza un documento XML e un file di trasformazione per generare un pacchetto dati, e poi assegnarlo a un dataset client:

```
XMLTransform1->SourceXMLFile = "CustomerDocument.xml";
XMLTransform1->TransformationFile = "CustXMLToCustTable.xtr";
ClientDataSet1->XMLData = XMLTransform1->Data;
```

Conversione dei nodi definiti dall'utente

Quando si definisce una trasformazione utilizzando xmlmapper.exe, è possibile specificare che alcuni dei nodi nel documento XML sono "User-defined ". I nodi definiti dall'utente sono nodi di cui si desidera fornire la trasformazione in codice, invece di contare su una traduzione diretta valore-del-nodo-valore-del-campo.

Utilizzando l'evento *OnTranslate* si può fornire il codice per tradurre i nodi definiti dall'utente. *OnTranslate* viene chiamato ogni volta che il componente *TXMLTransform* incontra un nodo definito dall'utente nel documento XML. Nel gestore di evento *OnTranslate*, è possibile leggere il documento di origine e specificare il valore risultante del campo nel pacchetto dati.

Ad esempio, il seguente gestore di evento *OnTranslate* converte un nodo nel documento XML con questo formato

```
<FullName>
   <Title> </Title>
   <FirstName> </FirstName>
   <LastName> </LastName>
</FullName>
```

in un singolo valore del campo:

```
void __fastcall TForm1::XMLTransform1Translate(TObject *Sender, AnsiString Id,
  _di_IDOMNode SrcNode, AnsiString &Value, _di_IDOMNode DestNode)
{
  if (Id == "FullName")
  {
    Value = "";
    if (SrcNode.hasChildNodes)
    {
      _di_IXMLDOMNode CurNode = SrcNode.firstChild;
      Value = SrcValue + AnsiString(CurNode.nodeValue);
      while (CurNode != SrcNode.lastChild)
      {
        CurNode = CurNode.nextSibling;
        Value = Value + AnsiString(" ");
        Value = Value + AnsiString(CurNode.nodeValue);
      }
    }
  }
}
```

Impiego di un documento XML come origine di un provider

Il componente *TXMLTransformProvider* permette di utilizzare un documento XML come se fosse una tabella di database. *TXMLTransformProvider* assembla i dati di un documento XML e gli applica gli aggiornamenti provenienti dai client. Ai client, come dataset client o ai broker XML appare come qualsiasi altro componente provider. Per informazioni sui componenti provider, consultare il Capitolo 28, "Uso di componenti provider". Per informazioni sull'utilizzo di componenti provider con i dataset client, consultare "Uso di un dataset client con un provider di dati" a pagina 27-26.

Mediante la proprietà *XMLDataFile* è possibile specificare il documento XML che dal quale il provider XML fornisce dati e al quale applica gli aggiornamenti.

I componenti *TXMLTransformProvider* utilizzano componenti interni *TXMLTransform* per la traduzione tra i pacchetti dati e il documento XML di origine: uno per tradurre il documento XML in pacchetti dati e uno per ritradurre i pacchetti dati nel formato XML del documento di origine dopo avere applicato gli aggiornamenti. Ai due componenti *TXMLTransform* si accede tramite le proprietà *TransformRead* e *TransformWrite*, rispettivamente.

Quando si utilizza *TXMLTransformProvider*, è necessario specificare le trasformazioni che i due componenti *TXMLTransform* utilizzano per tradurre tra pacchetti dati e il documento XML di origine. Si fa questo impostando la proprietà *TransformationFile* o

TransformationDocument del componente *TXMLTransform*, proprio come quando si utilizza un componente indipendente *TXMLTransform*.

Inoltre, se la trasformazione comprende anche nodi definiti dall'utente, bisogna fornire un gestore di evento *OnTranslate* ai componenti interni *TXMLTransform*.

Non è necessario specificare il documento di origine nei componenti *TXMLTransform* che sono i valori di *TransformRead* e di *TransformWrite*. Per *TransformRead*, l'origine è il file specificato dalla proprietà *XMLDataFile* del provider(benché, se si imposta *XMLDataFile* a una stringa vuota, sia possibile fornire il documento di origine utilizzando *TransformRead.XmlSource* o *TransformRead-XmlSourceDocument*). Per *TransformWrite*, l'origine è generata internamente dal provider quando applica gli aggiornamenti.

Utilizzo di documenti XML come client di un provider

Il componente *TXMLTransformClient* agisce come adattatore per consentire di usare un documento (o un insieme di documenti) XML come client di un application server remoto (o semplicemente come client di un dataset a cui si collega tramite un componente *TDataSetProvider*). Cioè, il client *TXMLTransform* consente di pubblicare i dati di database sotto forma di documento XML e di utilizzare le richieste di aggiornamento (inserimenti o cancellazioni) provenienti da un'applicazione esterna che li fornisce sotto forma di documenti XML.

Per specificare il provider dal quale l'oggetto *TXMLTransformClient* preleva i dati e al quale applica gli aggiornamenti, si imposta la proprietà *ProviderName*. Come con la proprietà *ProviderName* di un dataset client, *ProviderName* può essere il nome di un provider su un application server remoto, oppure un provider locale con lo stesso formato o lo stesso modulo dati dell'oggetto *TXMLTransformClient*. Per informazioni sui provider, consultare il Capitolo 28, "Uso di componenti provider".

Se il provider si trova su un application server remoto, è necessario utilizzare un componente connection DataSnap per collegarsi all'application server in questione. Si specifica il componente connection mediante la proprietà *RemoteServer*. Per informazioni sui componenti di connessione DataSnap, consultare "Connessione all'application server" a pagina 29-24.

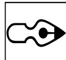
Prelievo di un documento XML da un provider

TXMLTransformClient utilizza un componente interno *TXMLTransform* per tradurre i pacchetti dati forniti dal provider in un documento XML. Si accede al componente *TXMLTransform* dandogli il valore della proprietà *TransformGetData*.

Prima di creare un documento XML che rappresenta i dati presi da un provider, è necessario specificare il file di trasformazione che *TransformGetData* utilizza per tradurre il pacchetto dati nel formato XML appropriato. Si fa questo impostando la proprietà *TransformationFile* o *TransformationDocument* del componente *TXMLTransform*, proprio come quando si utilizza un componente indipendente

TXMLTransform. Se la trasformazione include anche dei nodi definiti dall'utente, è necessariio fornire a *TransformGetData* anche un gestore di evento *OnTranslate*.

Non serve specificare il documento di origine di *TransformGetData*, dato che *TXMLTransformClient* lo preleva dal provider. Tuttavia, se il provider attende dei parametri di input, è bene impostarli prima di prelevare i dati. Per fornire questi parametri di input prima di prelevare dati dal provider si usa il metodo *SetParams*. *SetParams* prende due argomenti: una stringa XML da cui estrarre valori del parametro e il nome di un file di trasformazione per tradurre il file XML in un pacchetto dati. *SetParams* utilizza il file di trasformazione per convertire la stringa di XML in un pacchetto dati, e quindi dal pacchetto dati estrae i valori del parametro.



Se si desidera specificare in un altro modo il documento di parametro o la trasformazione, è possibile ridefinire ambedue gli argomenti. Basta impostare una delle proprietà sulla proprietà *TransformSetParams* per indicare il documento che contiene i parametri, o la trasformazione da utilizzare per la conversione, e poi quando si chiama *SetParams* impostare l'argomento che si desidera ridefinire a una stringa vuota. Per i dettagli, vedere "Conversione di documenti XML in pacchetti dati" a pagina 30-6.

Una volta configurato *TransformGetData* e forniti gli eventuali parametri di input, è possibile chiamare il metodo *GetDataAsXml* per prelevare lo XML. *GetDataAsXml* invia i valori dei parametri correnti al provider, preleva un pacchetto dati, lo converte in un documento XML e restituisce il documento come stringa. Si può salvare questa stringa in un file:

```
XMLTransformClient1->ProviderName = "Provider1";
XMLTransformClient1->TransformGetData->TransformationFile = "CustTableToCustXML.xtr";
XMLTransformClient1->TransFormSetParams->SourceXmlFile = "InputParams.xml";
XMLTransformClient1->SetParams("", "InputParamsToDP.xtr");
AnsiString XML = XMLTransformClient1->GetDataAsXml();
TFileStream pXMLDoc = new TFileStream("Customers.xml", fmCreate || fmOpenWrite);
__try
{
  pXMLDoc->Write(XML.c_str(), XML.Length());
}
__finally
{
  delete pXMLDoc;
}
```

Applicazione degli aggiornamenti da un documento XML a un provider

TXMLTransformClient permette anche di inserire tutti i dati tratti da un documento XML nel dataset del provider o di cancellare tutti i record di un documento XML dal dataset del provider. Per eseguire questi aggiornamenti, si chiama il metodo *ApplyUpdates*, passando

- Una stringa il cui valore è il contenuto del documento XML con i dati da inserire o da cancellare.

- Il nome di un file di trasformazione che può convertire i dati XML in un pacchetto delta di inserimento o di cancellazione. (Quando si definisce il file di trasformazione utilizzando XML mapper, si specifica se la trasformazione è per inserimento, oppure si cancella pacchetto delta.)
- Il numero di errori di aggiornamento ammessi prima di interrompere l'operazione di aggiornamento. Se il numero di record che non è stato possibile inserire o cancellare è inferiore al numero specificato, *ApplyUpdates* restituisce il numero di errori effettivi. Se il numero di record che non è stato possibile inserire o cancellare è superiore al numero specificato, l'operazione viene annullata e nessun aggiornamento viene eseguito.

La seguente chiamata trasforma il documento XML Customers.xml in un pacchetto delta e applica tutti gli aggiornamenti indipendentemente dal numero di errori:

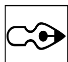
```
StringList1->LoadFromFile("Customers.xml");
nErrors = ApplyUpdates(StringList1->Text, "CustXMLToInsert.xtr", -1);
```

Parte

III

Creazione di applicazioni Internet

I capitoli inclusi nella sezione “Creazione di applicazioni Internet” presentano i concetti fondamentali e le conoscenze necessarie per la creazione di applicazioni da distribuire in Internet.



Non tutti i componenti descritti in questa sezione sono disponibili in tutte le versioni di C++Builder.

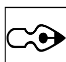
Capitolo

31

Scrittura di applicazioni CORBA

CORBA (Common Object Request Broker Architecture) è una specifica adottata da OMG (Object Management Group) per affrontare la complessità di sviluppo di applicazioni distribuite orientate agli oggetti.

Come dice il nome, CORBA fornisce un approccio orientato agli oggetti per la scrittura di applicazioni distribuite. Questo approccio è l'opposto di quello orientato ai messaggi descritto per le applicazioni HTTP nel Capitolo 32, "Creazione di applicazioni server per Internet". Sotto Corba, le applicazioni server implementano oggetti che possono essere usati da postazione remota dalle applicazioni client, tramite interfacce ben definite.



COM fornisce un altro approccio orientato agli oggetti per le applicazioni distribuite. Per maggiori informazioni su COM, vedere il Capitolo 38, "Panoramica sulle tecnologie COM". A differenza di COM, però, CORBA è uno standard che viene impiegato in piattaforme diverse da Windows. Questo vuol dire che è possibile scrivere client o server CORBA con C++Builder, che sono in grado di comunicare con applicazioni abilitate per CORBA in esecuzione su altre piattaforme.

La specifica CORBA definisce il modo in cui le applicazioni client comunicano con gli oggetti che sono implementati su un server. Questa comunicazione è gestita da un Object Request Broker (ORB). C++Builder è fortemente integrato con VisiBroker ORB di Borland (versione 4.5) in modo da semplificare lo sviluppo in CORBA.

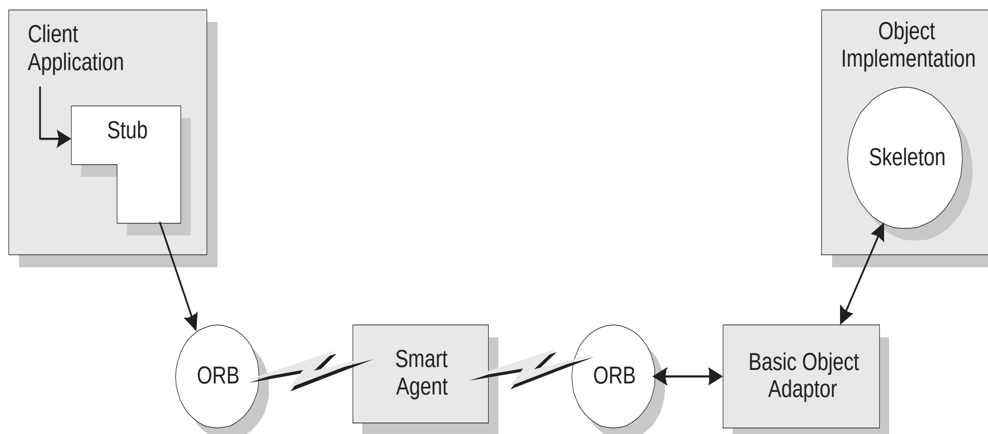
Oltre alla tecnologia base ORB, che abilita i client a comunicare con gli oggetti su macchine server, lo standard CORBA definisce un certo numero di servizi standard. Poiché questi servizi usano interfacce ben definite, gli sviluppatori possono scrivere client che li usano anche se i server sono scritti da altri produttori.

Panoramica di un'applicazione CORBA

Se si sta già programmando in modo orientato agli oggetti, si potrà notare che CORBA semplifica notevolmente la scrittura di applicazioni distribuite, in quanto

consente di usare oggetti remoti come se fossero locali. Ciò è dovuto al fatto che la progettazione di un'applicazione CORBA è molto simile a quella di una qualsiasi altra applicazione orientata agli oggetti, tranne per il fatto che include un layer aggiuntivo per la gestione della comunicazione in rete quando un oggetto risiede su un'altra macchina. Questo layer viene gestito da speciali oggetti chiamati stub e skeleton.

Figura 31.1 Struttura di un'applicazione CORBA



Sui client CORBA, lo stub agisce come un proxy per un oggetto che potrebbe essere implementato dallo stesso processo, da un altro processo o su un'altra macchina (server). Il client interagisce secondo le stesse modalità adottate con un qualunque altro oggetto.

Tuttavia, a differenza della maggior parte di oggetti, lo stub gestisce le chiamate d'interfaccia chiamando il software VisiBroker ORB installato sulla macchina client. Questo software usa uno Smart Agent (osagent) che viene eseguito in un punto della rete locale. Questo agent è un servizio dinamico di directory distribuite che individua un server disponibile che fornisce l'implementazione vera e propria dell'oggetto.

Sul server CORBA il software ORB passa le chiamate d'interfaccia a un skeleton generato automaticamente. Questo comunica con il software ORB tramite il Basic Object Adaptor (BOA). Usando il BOA, lo skeleton registra l'oggetto con lo Smart Agent, ne indica il campo d'azione (specifica se può essere usato su macchine remote) e segnala quando gli oggetti sono istanziati e pronti per rispondere ai client.

Gli stub e gli skeleton

La base di un oggetto distribuito sotto CORBA è l' interfaccia. L'interfaccia è molto simile a una definizione di classe, tranne per il fatto che non include informazioni sull'implementazione. Le interfacce vengono definite utilizzando il linguaggio di definizione delle interfacce di CORBA (IDL). La definizione dell'interfaccia può quindi essere aggiunta al progetto sotto forma di file IDL. Per maggiori informazioni sui file IDL, stub e skeleton consultare il *manuale di VisiBroker Programmer*. Per

maggiori informazioni sulle operazioni coi file IDL in C++Builder, consultare il paragrafo "Definizione delle interfacce dell'oggetto" a pagina 31-5.

Quando si compila il file IDL, C++Builder genera automaticamente due file .cpp. Il primo, il file client, contiene l'implementazione della classe stub. L'altro, il file server, contiene l'implementazione della classe skeleton. Stub e skeleton forniscono il meccanismo che consente alle applicazioni CORBA di coordinare le chiamate di interfaccia. Il meccanismo di marshaling

- Prende un'istanza di oggetto nel processo del server e lo rende disponibile al codice nel processo del client.
- Trasferisce gli argomenti di una chiamata d'interfaccia passati dal client e li colloca nello spazio di processo dell'oggetto remoto.

Quando l'applicazione client chiama il metodo di un oggetto CORBA, spinge gli argomenti nello stack e chiama oggetto stub. Lo stub scrive gli argomento in un buffer di smistamento e trasmette la chiamata in una struttura dell'oggetto remoto.Lo skeleton del server apre questa struttura, spinge gli argomenti sullo stack e chiama l'implementazione dell'oggetto. In sostanza, lo skeleton ricrea la chiamata del client nel proprio spazio d'indirizzo.

Utilizzo di Smart Agent

Lo Smart Agent (osagent) è un servizio dinamico distribuito di directory che individua un server disponibile che implementa un oggetto. Se si può scegliere tra più server, lo Smart Agent fornisce il bilanciamento del carico. Questo servizio protegge anche dai guasti del server tentando di riavviarlo quando una connessione non riesce, oppure, se necessario, individuando un server su un altro host.

Uno Smart Agent deve essere avviato su almeno un host della rete locale, nel punto in cui la rete fa riferimento ad un'altra rete all'interno della quale può essere inviato un messaggio di trasmissione. Il software ORB individua uno Smart Agent utilizzando un messaggio di trasmissione. Se il la rete include più Smart Agent, la tecnologia ORB usa quello che risponde per primo. Una volta individuato lo Smart Agent, il software ORB usa un protocollo UDP punto a punto per comunicare con lo Smart Agent. Questo protocollo consuma meno risorse di rete di una connessione TCP.

Quando una rete include più Smart Agent, ciascuno di essi riconosce un sottogruppo di oggetti disponibili, e comunica con gli altri Smart Agent per individuare gli oggetti che non è in grado di riconoscere direttamente. Se uno Smart Agent termina inaspettatamente, gli oggetti dei quali tiene la traccia vengono registrati automaticamente di nuovo con un altro Smart Agent disponibile.

Per informazioni sulla configurazione e sull'uso degli Smart Agent sulle reti locali, consultare il manuale *VisiBroker Installation and Administration Guide*.

Attivazione delle applicazioni server

Quando l'applicazione server viene avviata, essa informa l' ORB (tramite il Basic Object Adaptor) degli oggetti che possono accettare le chiamate del client. Questo codice per inizializzare l'ORB e per informarlo del fatto che il server è attivo e pronto, viene aggiunto automaticamente all'applicazione dal wizard usato per avviare l'applicazione server CORBA.

Normalmente, le applicazioni server CORBA vengono avviate manualmente. Però, è possibile usare l'Object Activation Daemon (OAD) per avviare i server o istanziare i loro oggetti solo quando i client hanno bisogno di usarli.

Per usare l'OAD, occorre registrarvi i propri oggetti. La registrazione di oggetti con l'OAD memorizza le associazioni tra gli oggetti stessi e l'applicazione server che li implementa in un database chiamato Implementation Repository.

Una volta che nell'Implementation Repository c'è una voce per l'oggetto, l'OAD simula l'applicazione per l'ORB. Quando un client richiede l'oggetto, l'ORB contatta l'OAD come se fosse l'applicazione server. L'OAD quindi evade la richiesta del client per il server vero e proprio, lanciando se necessario l'applicazione.

Per informazioni dettagliate sulla registrazione degli oggetti con l'OAD, consultare il *VisiBroker Programmer's Guide*.

Binding dinamico delle chiamate alle interfacce

Normalmente, i client CORBA usano un binding statico per chiamare le interfacce degli oggetti sul server. Questo approccio ha molti vantaggi, che comprendono prestazioni più rapide e il controllo del tipo in fase di compilazione.Tuttavia, ci sono casi in cui si può conoscere solo in fase di esecuzione quale interfaccia si intende usare. A volte, tuttavia, può succedere che non si sappia quale oggetto o interfaccia utilizzare fino al momento dell'esecuzione. In questi casi, C++Builder permette di eseguire il bind delle interfacce dinamicamente in fase di esecuzione.

Se si usa un binding dinamico, esso risulta d'aiuto nel registrare le interfacce con l'Interface Repository.

Per maggiori informazioni su come utilizzare il binding dinamico nelle applicazioni client CORBA, consultare la sezione "Dynamic Invocation Interface (DII)" nel manuale *VisiBroker Programmer's Guide*.

Scrittura di server CORBA

C++Builder include diversi wizard per semplificare il processo di sviluppo di server CORBA. I passi seguenti descrivono come creare un server CORBA utilizzando C++Builder:

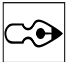
1 Definire le interfacce dell'oggetto. Queste interfacce definiscono il modo in cui le applicazioni client interagiscono col server. Inoltre, esse rappresentano la base da cui C++Builder crea le implementazioni di stub e skeleton.

2 Utilizzare il wizard CORBA server per creare una nuova applicazione server CORBA che includa il codice per inizializzare all'avvio BOA e ORB di CORBA.
3 Compilare i file IDL che contengono le definizioni dell'interfaccia in classi skeleton (e classi stub).
4 Utilizzare il wizard CORBA object per definire (e istanziare) le classi dell'implementazione.
5 Implementare gli oggetti CORBA completando le classi create nel passo 5.
6 Se necessario, modificare le interfacce CORBA e includere tali modifiche nelle classi dell'implementazione.

Oltre ai passi elencati in precedenza, si può scegliere di registrare i file IDL con l'Interface Repository e Object Activation Daemon.

Definizione delle interfacce dell'oggetto

Le interfacce degli oggetti CORBA vengono definite utilizzando il linguaggio di definizione delle interfacce di CORBA (IDL). IDL ha una sintassi simile a quella del C++, tanto che un file IDL è molto simile a un file header di C++. Il file IDL si comporta anche in odo molto simile ad un file header, dichiarando le interfacce che possono essere condivise, proprio come un file header dichiara le classi che possono essere condivise. In CORBA, comunque, le interfacce (le classi) vengono condivise con le altre applicazioni piuttosto che con (oppure in aggiunta ad) altri moduli nella stessa applicazione.



Il termine IDL è utilizzato in contesti differenti per fare riferimento a linguaggi di definizione di interfacce simile ma non identici. Esiste un IDL CORBA (definito da OMG), un IDL Microsoft (utilizzato per COM) e un IDL DCE. Per maggiori informazioni sull'IDL CORBA, consultare la Guida in linea.

Per definire un nuovo file IDL dall'interno di C++Builder, scegliere File | New | Other e selezionare dalla pagina Multitier della finestra di dialogo New Items l'opzione CORBA IDL File. Verrà aperto l'editor del codice con un file IDL vuoto e il file IDL verrà aggiunto al progetto corrente.

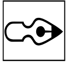
Se esiste già un file IDL che definisce gli oggetti, è possibile semplicemente aggiungere questo file al progetto scegliendo Project | Add to Project, selezionando IDL files come tipo di file e selezionando quindi il file IDL esistente.

Utilizzo del CORBA Server Wizard

Per cominciare un nuovo progetto CORBA Server, scegliere l'opzione File | New | Other e, dalla pagina di Multitier della finestra di dialogo di New Items, scegliere l'icona identificata come CORBA Server. Il wizard CORBA Server permette di indicare se si desidera creare un'applicazione per console oppure un'applicazione per Windows.

Se si sta creando un'applicazione per console, è possibile specificare se il server utilizzerà le classi della VCL. Se non si seleziona la casella di controllo VCL, tutto il codice generato può essere portato su altre piattaforme.

Oltre a scegliere il tipo di applicazione, è possibile includere nel progetto qualsiasi file IDL esistente, oppure è possibile specificare che si desidera includere un nuovo file IDL vuoto. Il progetto del server eventualmente deve includere uno o più file IDL che definiscono le interfacce utilizzate dai client per comunicare col server.



ISe non si aggiungono i file IDL all'inizio di un progetto per server CORBA, è sempre possibile aggiungerli in seguito scegliendo l'opzione Project | Add to Project (per i file IDL e esistenti) oppure scegliendo dalla pagina Multitier della finestra di dialogo New Items l'opzione CORBA IDL file (per definire un nuovo file IDL).

Una volta indicato il tipo di server desiderato e aver fatto clic su OK, il wizard CORBA Server creerà un nuovo progetto server del tipo specificato, aggiungendo al file di progetto le librerie CORBA e il codice di avvio per inizializzare ORB (Object Request Broker) e il BOA (Basic Object Adaptor).

Il codice generato in automatico dichiara due variabili, orb e boa che è possibile utilizzare per comunicare con ORB e BOA:

```
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
CORBA::BOA_var boa = orb->BOA_init(argc, argv);
```

Se è stata specificata un'applicazione per console, si vedrà anche la linea seguente:

```
boa->impl_is_ready();
```

Questa chiamata consente all'applicazione di ricevere messaggi dal BOA, in modo molto simile a come un loop di messaggi di Windows consente a un'applicazione Windows di ricevere i messaggi da Windows. (Le applicazioni Windows utilizzano il loop di messaggi di Windows anche per i messaggi CORBA).

Per maggiori informazioni su ORB e BOA, consultate la documentazione di VisiBroker.

Generazione di stub e skeleton da un file IDL

Dopo aver aggiunto a un progetto CORBA il file IDL che descrive le interfacce CORBA, è possibile generare le classi stub e skeleton che gestiscono la comunicazione tra il server e le sue applicazioni client.

Per generare le classi stub e skeleton, è sufficiente compilare il file IDL. È possibile compilare questo file scegliendo l'opzione Project | Compile Unit quando il file IDL è visualizzato nell'editor di codice, oppure fare clic destro sul file IDL nel project manager e scegliere Compile. Il compilatore IDL genera due nuovi file che appaiono nell'editor di codice e nel project manager. Questi sono

- Il file del server (xxx s.cpp). Questo file contiene l'implementazione delle classi skeleton.
- Il file client (xxx c.cpp). Questo file contiene l'implementazione delle classi stub.

La pagina CORBA della finestra di dialogo Project Options consente di influenzare il modo in cui vengono generate a partire dal file IDL le classi stub e skeleton. Ad esempio, è possibile specificare che si desidera includere nel progetto solo la unit generata per il server, oppure specificare che si vogliono includere le classi Tie solo se il server utilizza il modello della delega (descritto nella sezione “Uso del modello di delega” a pagina 31-8). Queste opzioni influenzano la compilazione di tutti i file IDL inclusi nel progetto corrente.

Uso del Wizard CORBA Object Implementation

Per implementare gli oggetti CORBA definiti nel file IDL, si devono creare classi di implementazione che corrispondano a ogni classe skeleton. Per fare questo, scegliere l’opzione File | New | Other, e dalla pagina Multitier della finestra di dialogo New Items, scegliete l’opzione CORBA Object Implementation. In tal modo verrà attivato il wizard CORBA Object Implementation.

Il wizard CORBA Object Implementation elenca tutte le interfacce definite nei file IDL inclusi nel progetto. Scegliere un file IDL e, a partire da quel file IDL, selezionare l’interfaccia che si desidera implementare. Assegnare un nome alla classe di implementazione e specificare il nome della unit dei file .h e .cpp che conterranno la definizione e l’implementazione dell’oggetto.

Nel Wizard, indicare se le classi di implementazione dovrebbero essere discendenti delle classi skeleton generate in automatico, o se si sta utilizzando un modello di delega (classi tie). Se si sta utilizzando il modello di delega (e l’applicazione server utilizza la VCL), è anche possibile chiedere al wizard di creare un modulo dati per le classi dell’implementazione. Ciò consente di aggiungere i componenti, che si potrebbero voler utilizzare durante la scrittura dell’implementazione, al modulo dati della stessa.

È possibile richiedere al wizard di aggiungere del codice per istanziare gli oggetti CORBA all’avvio dell’applicazione, così da renderli disponibili a ricevere le richieste dei client. Se si istanziano gli oggetti in fase di avvio, è necessario assegnar loro un nome in modo che i possano essere localizzati dai client. Questo nome viene passato al costruttore come parametro, e non è il nome della variabile che referenzia l’istanza. Quando le applicazioni client eseguono il bind all’oggetto CORBA, utilizzano questo nome per indicare l’oggetto desiderato.

Facendo clic su OK, il wizard CORBA Object Implementation genera la definizione per la classe dell’implementazione e genera un’implementazione in cui ogni metodo ha un corpo vuoto. Inoltre, può aggiungere anche del codice per istanziare gli oggetti, nel caso questa possibilità sia stata indicata.

Prima che queste variazione vengano aggiunte al progetto, è possibile rivederle e modificarle. Per fare ciò, prima di fare clic su OK, selezionare la casella Show Updates. Dopo avere rivisto le variazioni e averle eventualmente modificate, esse vengono aggiunte al progetto. Se non si sceglie di rivedere e modificare le variazioni all’uscita dal wizard, sarà possibile modificarle in seguito nell’editor del codice.

Per implementare più interfacce, è necessario eseguire varie volte il wizard CORBA Object Implementation, una volta per ciascuna interfaccia. Tuttavia, è possibile

richiedere al wizard di generare più istanze di una singola classe di implementazione fornendo più nomi.

Istanze di oggetti CORBA

Per ogni oggetto CORBA, è necessario decidere come dovrà essere istanziato. Esistono due possibilità:

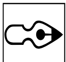
- L'oggetto viene istanziato all'avvio dell'applicazione server. In questo caso, l'oggetto è disponibile alle applicazioni client ad ogni richiesta fatta al server.
- L'oggetto viene istanziato in risposta a una richiesta del client (una chiamata a un metodo di un oggetto esistente). In questo caso, l'applicazione server istanzia l'oggetto all'interno dell'implementazione di un metodo di interfaccia, e restituisce un riferimento all'oggetto da quel metodo.

Se l'oggetto viene istanziato all'avvio dell'applicazione, selezionare la casella Istantiate in main nel wizard CORBA Object. Il wizard aggiunge al file della sorgente del progetto il codice necessario per istanziare direttamente la classe dell'implementazione. È possibile esaminare questo codice scegliendo l'opzioneProject|View Source. Dopo le linee inserite dal Wizard CORBA Server per chiamare ORB_init e BOA_init e prima della chiamata impl_is_ready(), è possibile trovare il codice che istanzia la classe di implementazione e informa il boa che l'oggetto può accettare le richieste:

```
MyObjImpl MyObject_TheObject("TheObject"); // create the object instance
boa->obj_is_ready(&MyObject_TheObject); // inform the boa it can take requests
```

È possibile aggiungere codice di inizializzazione tra il costruttore e la chiamata a obj_is_ready:

```
MyObjImpl MyObject_TheObject("TheObject"); // create the object instance
MyObject_TheObject.Initialize(50000);
boa->obj_is_ready(&MyObject_TheObject); // inform the boa it can take requests
```


Se si utilizzano le classi tie (vedere di seguito), sarà presente anche una chiamata per istanziare la classe tie.

Se gli oggetti non vengono istanziati all'avvio, si può chiamare il costruttore (e, se necessario, il costruttore della classe tie) da un metodo e ottenere un riferimento all'istanza dell'oggetto. È buona norma aggiungere anche una chiamata al metodo obj_is_ready del BOA's.

Agli oggetti che accettano chiamate da client CORBA occorre assegnare un nome, altrimenti i client non saranno in grado di trovarli. Agli oggetti transitori (oggetti creati in risposta a chiamate del client e restituiti come riferimenti) non è necessario assegnare un nome.

Uso del modello di delega

Per impostazione predefinita, gli oggetti CORBA sono discendenti di classi skeleton create al momento della compilazione di un file IDL. Ciò significa che le classi dell'oggetto ereditano tutto il codice che coordina le chiamate all'interfaccia e interagisce col BOA. Tuttavia, se si desidera esporre oggetti nel server CORBA già scritti in precedenza, oppure derivati da classi della VCL (che non consente l'eredità

multipla), o che si vogliono condividere con altre applicazioni non-CORBA, è possibile utilizzare classi server che non dipendono da CORBA oppure i file server generati.

L'applicazione deve utilizzare un modello di delega per esporre gli oggetti che non discendono da una classe skeleton. Un modello della delega è un modello in cui le istanze delle classi skeleton non implementano direttamente gli oggetti CORBA, ma piuttosto passano una classe di interfaccia a una classe di implementazione completamente separata. Ciò significa che la classe di implementazione non è un discendente di skeleton, ma piuttosto che viene chiamato dallo skeleton.

Per utilizzare il modello di delega, si devono generare le classi tie al momento della compilazione dei file IDL. Prima di compilare i file IDL, scegliere l'opzione Project | Options, e sulla pagina IDL della finestra di dialogo di Project Options, selezionare la casella per generare le classi tie.

Una volta compilato il file IDL, oltre alla classe skeleton, il file del server contiene una classe tie speciale. Questa classe tie agisce come un proxy per la classe dell'implementazione, e, relativamente alle implementazioni di metodi, essa delega la (passa la chiamata alla) classe dell'implementazione.

Quando si utilizza il wizard CORBA Object per definire la classe dell'implementazione, scegliere il pulsante di opzione Delegation (Tie), in modo tale che la classe dell'implementazione non venga creata come un discendente della classe skeleton. Il wizard CORBA Object aggiunge un commento accanto alla dichiarazione della classe. Non rimuovere questo commento: è utilizzato da C++Builder per identificare la classe come una classe di implementazione di CORBA.

Oltre a cambiare il modo in cui il wizard CORBA Object genera la classe dell'implementazione, la scelta del pulsante Tie influenza anche tutto il codice generato in automatico che istanzia l'oggetto.

Quando l'applicazione server istanzia la classe dell'implementazione, deve istanziare immediatamente dopo la classe tie associata, passando al costruttore l'istanza della classe di implementazione sotto forma di argomento. Se la classe dell'implementazione viene istanziata all'avvio, il codice necessario viene generato automaticamente dal wizard CORBA Object. Tuttavia, se si sta istanziando dinamicamente l'oggetto CORBA in fase di esecuzione, si deve aggiungere questo codice manualmente. Per esempio:

```
MyObjImpl myobj(); // instantiate implementation object
_tie_MyObj<MyObjImpl> tieobj(myobj, "InstanceName");
```

Visualizzazione e modifica dei cambiamenti

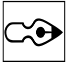
Prima di aggiungere qualsiasi modifica al progetto, è buona norma esaminare ed eventualmente variare tali modifiche. Questa operazione ha due scopi:

- Prendere atto di tutte le modifiche apportate al progetto.Non verrà aggiunto alcun codice creato in automatico che potrebbe generare un comportamento di cui non si è a conoscenza.

- È possibile personalizzare le modifiche senza doverle andare a cercare in tutti i file dell'implementazione. Ciò risulta particolarmente utile se si stanno aggiungendo delle modifiche a un file che contiene già molto codice.

Per esaminare il codice e apportare eventuali modifiche, utilizzare la finestra di dialogo Project Updates. Questa finestra di dialogo viene visualizzata quando si seleziona l'opzione Show Updates in un qualsiasi wizard CORBA oppure quando si sceglie il comando Edit|CORBA Refresh.

La finestra di dialogo Project Updates elenca le modifiche in un pannello sulla sinistra. È possibile rifiutare tutte le modifiche deselezionando la casella che appare alla sua sinistra. Se si rifiuta una modifica, questa non viene inserita nel progetto, e sarà necessario scrivere manualmente il codice corrispondente.



Le modifiche sono elencate nell'ordine in cui sono state fatte. Eliminando una modifica si potrebbero rimuovere altre modifiche che dipendono da essa. Per esempio, deselezionando la creazione di un nuovo file, verranno deselezionate le modifiche che rappresentano il codice aggiunto a quel file.

Man mano che si scorre l'elenco di modifiche, si vedrà il codice aggiunto nel pannello del codice. Il file che è stato modificato per contenere quel codice viene elencato nella parte inferiore del pannello del codice.

Se si desidera personalizzare una modifica è possibile modificare il pannello del codice. Per esempio, si potrebbe voler aggiungere una ulteriore classe antenato alla classe dell'implementazione, in modo che erediti il comportamento di una delle classi esistenti.Si potrebbe anche aggiungere del codice per implementare i metodi, compilando i corpi dei metodi vuoti generati in automatico.

Terminata la revisione e l'eventuale modifica dei cambiamenti generati in automatico, fare clic su OK per aggiungerli al progetto e includere anche le modifiche apportate. Se non desidera accettare alcuna modifica, annullare la finestra di dialogo.

Implementazione degli oggetti CORBA

Se non si sta utilizzando un modello di delega, la classe dell'implementazione è un discendente della classe skeleton generata all'atto della compilazione del file IDL. In caso contrario, non esistono antenati espliciti alle classi dell'implementazione. È possibile modificare questa definizione della classe in modo che erediti da un'altra classe nell'applicazione server, ma non si devono rimuove eventuali eredità esistenti da una classe skeleton.

Si esamini la seguente dichiarazione estratta da un file IDL, account.idl:

```
interface Account {
  float balance();
};
```

Quando si compila il file IDL (senza le classi tie), l'header del server generato (account_s.hh) include la dichiarazione di una classe skeleton con metodi virtuali puri per ogni metodo dell'interfaccia.

Il wizard CORBA Object crea una unit dell'implementazione che deriva una classe di implementazione dalla classe skeleton (_sk_Account). Il file header assomiglia al seguente:

```
#ifndef Interface1ServerH
#define Interface1ServerH
#include "account_s.hh"
//---------------------------------------------------------------------------
class AccountImpl: public _sk_Account
{
  protected:
  public:
    AccountImpl(const char *object_name=NULL);
    CORBA::float balance();
};

#endif
```

Si potrebbero aggiungere a questa definizione di classe altri membri dati o metodi potrebbero occorre per l'implementazione. Per esempio:

```
#ifndef Interface1ServerH
#define Interface1ServerH
#include "account_s.hh"
//---------------------------------------------------------------------------
class AccountImpl: public _sk_Account
{
  protected:
    CORBA::float _bal;
  public:
    void Initialize(CORBA::float startbal); // not available to clients
    AccountImpl(const char *object_name=NULL);
    CORBA::float balance();
};

#endif
```

Questi metodi e proprietà aggiuntivi sono rese disponibili dall'applicazione server ma non accessibili dai client.

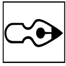
Completare nel file .cpp generato il corpo principale della classe dell'implementazione, in modo da ottenere un oggetto CORBA funzionante:

```
AccountImpl::AccountImpl(const char *object_name):
    _sk_Account(object_name)
{
}

CORBA::float AccountImpl::balance()
{
  return _bal;
};

void Initialize(CORBA::float startbal) // not available to clients
{
  _bal = startbal;
```

```
}
```



A partire dal server CORBA, è possibile scrivere del codice in grado di interagire col BOA. Per esempio, utilizzando il BOA, è possibile nascondere temporaneamente, o disattivare, gli oggetti server e successivamente riattivarli. Per maggiori informazioni su come scrivere del codice in grado di interagire col BOA, consultare il manuale *VisiBroker Programmer's Guide*.

Protezione da conflitti di thread

Per impostazione predefinita, le applicazioni CORBA sono multi-thread. Ciò significa che quando si implementano gli oggetti CORBA è necessario cautelarsi da eventuali conflitti di thread.

A meno che si specifichi altrimenti, il BOA unisce i thread client, il che significa che tali richieste dei client possono utilizzare qualsiasi thread disponibile. È possibile, tuttavia, fare in modo che ciascun client utilizzi sempre lo stesso thread avviando il BOA imponendo un certo numero di thread per sessione. Se ciascun client utilizza sempre lo stesso thread, è possibile memorizzare le informazioni persistenti del client in variabili di thread. Per informazioni su come impostare la politica di threading quando si avvia BOA, consultare il manuale *VisiBroker Programmer's Guide*.

La libreria VisiBroker include delle classi che possono essere d'aiuto nel proteggere il codice da eventuali conflitti di thread. Queste vengono definite nel file header vthread.h, installato nella directory dei file include di VisiBroker. Le classi di supporto per il thread di VisiBroker includono un mutex (VisMutex), una variabile di condizione (VISCondition) e un blocco in lettura/scrittura che funziona, più o meno, come il sincronizzatore multi-read/exclusive-write della VCL (VISRWLock). Uno dei benefici derivanti dall'utilizzo di queste classi è che sono portabili su tutte le piattaforme supportate da VisiBroker.

Per esempio, è possibile proteggere dati di istanza aggiungendo un campo VisMutex alla classe dell'implementazione:

```
class A {
  VISMutex _mtx;
  ...
}
```

In seguito, quando sarà necessario sincronizzare l'accesso a dati di istanza, sarà possibile bloccare il mutex dall'interno di un qualsiasi metodo che accede i dati di istanza:

```
void A::AccessSharedMemory(...)
{
  VISMutex_var lock(_mtx); // acquires a lock that is released on exit
// add code here to access the instance data
}
```

Occorre notare che il mutex viene rilasciato al termine dell'esecuzione di AccessSharedMemory, anche se viene sollevata un'eccezione dall'interno del corpo del metodo.

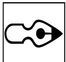
Se si sta utilizzando la VCL nell'applicazione server, è bene ricordare che C++Builder include varie classi che possono essere d'aiuto nella protezione contro eventuali

conflitti di thread. Per maggiori informazioni, consultare il Capitolo 11, "Scrittura di applicazioni multi-thread".

Modifica delle interfacce CORBA

Se si apportano delle modifiche alle interfacce nei file IDL dopo aver generato le classi dell'implementazione utilizzando il wizard CORBA Object, C++Builder consente di aggiornare automaticamente il progetto del server in modo da rispecchiare tali modifiche, senza perdere il lavoro già fatto scrivendo le classi dell'implementazione.

Dopo aver modificato i file IDL, scegliere il comando Edit|CORBA Refresh. Questo comando ricompila i file IDL così che il file del client e del server generati in automatico rispecchino le nuove definizioni dell'interfaccia e opzioni correnti del compilatore. Se i file IDL vengono compilati con successo, è possibile esaminare e modificare i cambiamenti apportati da C++Builder utilizzando la finestra di dialogo Project Updates.



Se C++Builder non trova l'interfaccia che corrisponde a una delle classi dell'implementazione esistenti, chiede se è stato cambiato nome all'interfaccia. Se si risponde di sì, verrà chiesto di indicare in che modo sono state rinominate le interfacce, in modo che le classi di implementazione possano essere fatte corrispondere correttamente alle interfacce.

Se si usa il comando Refresh di CORBA, alle classi di implementazione vengono aggiunti dei nuovi metodi e le dichiarazioni vengono aggiornate in modo da riflettere le modifiche apportate ai metodi e agli attributi esistenti. Tuttavia, alcune modifiche non provocano alcun aggiornamento. In particolare, se si cancella un metodo o un attributo, il relativo codice non viene cancellato. Invece, i metodi cancellati continuano a far parte della classe, ma non sono più disponibili ai client CORBA. Analogamente, se si cambia il nome a un metodo o a un attributo, questa modifica viene trattata come una cancellazione e un'aggiunta: il vecchio metodo o attributo viene conservato e viene generato del codice per un nuovo metodo o attributo.

Registrazione delle interfacce del server

Benché non sia necessario registrare le interfacce del server se si sta utilizzando il bindind statico delle chiamate dei client agli oggetti del server, la registrazione delle interfacce è un'operazione consigliata.Esistono due programmi di utilità che consentono di registrare le interfacce:

- L'**Interface Repository**. Se si registrano le interfacce con Interface Repository, i client possono ottenere da programma le informazioni sulle interfacce, se utilizzano l'interfaccia di chiamata dinamica (DII). Per maggiori informazioni sull'utilizzo della DII, consultare la sezione "Utilizzo dell'interfaccia di chiamata dinamica" a pagina 31-16. La registrazione mediante l'Interface Repository risulta anche essere un modo conveniente per consentire ad altri sviluppatori di vedere le interfacce durante la scrittura delle applicazioni client.

È possibile registrare le interfacce con l'Interface Repository scegliendo il comando Tools|IDL Repository.

- L'**Object Activation Daemon**. Effettuando la registrazione con l' Object Activation Daemon (OAD), non è necessario lanciare il server o istanziare gli oggetti finché non sono necessari ai client. In questo modo è possibile risparmiare risorse sul sistema server.

Scrittura di client CORBA

Quando si scrivete un client CORBA, il primo passo è quello di garantire che l'applicazione client possa parlare col software ORB sulla macchina client. Per fare ciò, utilizzare il wizard CORBA Client. Scegliere il comando File|New|Other e, dalla pagina Multitier della finestra di dialogo New Items, scegliete quindi l'icona identificata come CORBA Client. Il wizard CORBA Client consente di indicare se si vuole creare un'applicazione per console oppure un'applicazione per Windows.

In modo analogo a un'applicazione server CORBA, è possibile specificare se l'applicazione client CORBA utilizzerà le classi VCL. Se non si seleziona la casella di controllo VCL, tutto il codice generato può essere portato su altre piattaforme.

Nel wizard CORBA Client, includere tutti i file IDL che definiscono le interfacce degli oggetti del server che si vogliono utilizzare. Benché sia possibile creare un'applicazione client CORBA senza includere alcun file IDL (aggiungendo esplicitamente al progetto una unit client generata), questa non è la modalità preferita. Una volta che il progetto include i file IDL per le interfacce del server, è possibile utilizzare il wizard Use CORBA Object per eseguire il bind agli oggetti sul serverNote

Se i file IDL non vengono aggiunti quando si inizia un progetto di un client CORBA, è sempre possibile aggiungerli in seguito mediante il comando Project|Add to Project.

Il wizard CORBA Client crea sempre un nuovo progetto client del tipo specificato, aggiungendo le librerie CORBA al file di progetto e il seguente codice di avvio per inizializzare l'ORB (Object Request Broker).

```
CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
```

Se si ha intenzione di passare interfacce callback al server CORBA, sarà necessario inizializzare il BOA (Basic Object Adaptor) anche nell'applicazione client. Per fare ciò, è sufficiente selezionare nel wizard la casella di controllo opportuna.

Quindi, procedere con la scrittura dell'applicazione nello stesso modo in cui si scrive qualsiasi altra applicazione C++Builder. Comunque, quando si vorranno utilizzare oggetti che sono definiti nell'applicazione server, non si opererà direttamente con un'istanza dell'oggetto. Invece, si otterrà un riferimento a un oggetto CORBA e si lavorerà con quello. Il riferimento all'oggetto CORBA può essere ottenuto in due modi, a seconda che si desideri utilizzare il binding statico o dinamico.

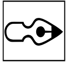
Per utilizzare il binding statico, è possibile richiamare il wizard Use CORBA Object scegliendo il comando Edit|Use CORBA Object. L'utilizzo del binding statico è più

rapido rispetto all'uso di quello dinamico, oltre a fornire ulteriori vantaggi quali il controllo sui tipi in fare di compilazione e il completamento del codice.

A volte, tuttavia, può succedere che non si sappia quale oggetto o interfaccia utilizzare fino al momento dell'esecuzione. In questi casi, si può utilizzare il binding dinamico. Il binding dinamico utilizza un oggetto CORBA generico che passa le richieste al server utilizzando uno tipo di CORBA particolare denominato Any.

Uso di stub

Le classi stub vengono generate automaticamente quando si compila il file IDL. Esse sono definite nei file client generati, i cui nomi assumono la forma NomeBase_c.cpp e NomeBase_c.hh.



Nella pagina CORBA della finestra di dialogo Project Options è possibile indicare a C++Builder di generare solamente i file del client (stub) e non i file del server.

Quando si scrive un client CORBA, non si modifica il codice nei file client generati. Si istanziano, invece, le classi stub nel punto in cui vengono utilizzate. Per fare ciò, scegliere Edit|Use CORBA Object per attivare il wizard Use CORBA Object.

Nel wizard Use CORBA Object, specificare il file IDL che contiene l'interfaccia a cui si desidera accedere e selezionare l'interfaccia che verrà utilizzata. Se si vuole eseguire il bind solo a una specifica istanza con un certo nome dell'oggetto CORBA, è possibile indicare, se si vuole, un nome per l'Oggetto CORBA.

Il wizard Use CORBA Object consente di scegliere fra vari meccanismi per eseguire il bind a un oggetto del server.

- Se l'applicazione client è un'applicazione Windows che utilizza la VCL, può creare una proprietà sulla scheda dell'applicazione per contenere un'istanza della classe stub dell'oggetto CORBA. È possibile utilizzare quindi questa proprietà come un'istanza dell'oggetto del server CORBA.
- Se si sta creando un'applicazione per console, il wizard può istanziare la classe stub come una variabile nella funzione main() dell'applicazione. In modo analogo, se si sta creando un'applicazione Windows, può istanziare la classe stub come una variabile nella funzione WinMain().
- Sia che si stia creando un'applicazione Windows o un'applicazione per console, il wizard può aggiungere una proprietà a una classe esistente in una qualsiasi unit specificata, oppure iniziare una nuova classe che include una proprietà per l'istanza dello stub.

Indipendentemente dal meccanismo scelto, il wizard aggiunge tutti i file header necessari e genera il codice che esegue il bind di una variabile stub o di una proprietà all'oggetto del server CORBA. Per esempio, il codice seguente istanzia lo stub per un'interfaccia del server, di nome MyServerObj nella funzione main() di un'applicazione per console:

```
#include <corba.h>
#include <condefs.h>
#include "MyServerObj_c.hh"
#pragma argsused
```

```
int main(int argc, char* argv[])
{
  try
  {
    // Initialize the ORB
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
    MyServerObj_var TheObject = MyServerObj::_bind("InstanceName");
  }
  catch(const CORBA::Exception& e)
  {
    cerr << e << endl;
    return(1);
  }
  return 0;
}
```



Se si copia il codice che esegue il bind a un oggetto del server in un'altra parte dell'applicazione, è bene accertarsi che la variabile stub sia una variabile locale oppure il membro dati di una classe. Le variabili stub globali sono abbastanza delicate in quanto non possono rilasciare il proprio contatore dei riferimenti di CORBA prima che venga rilasciata la variabile orb. Se si deve utilizzare necessariamente una variabile stub globale, assicurarsi di annullare il bind impostando la variabile a NULL prima di rilasciare l'orb.

Se l'applicazione client include solamente la unit client generata (BaseName_c), non è possibile utilizzare il wizard CORBA Object per eseguire il bind a oggetti del server. Tale codice dovrà essere generato dal programmatore. Si potrebbe anche voler scrivere del codice personale per eseguire il bind in modo da sfruttare le opzioni di bind di VisiBroker. (Per esempio, si potrebbe voler specificare un determinato server, disattivare la funzione che tenta automaticamente di rieseguire il bind a un server nel caso vada perda la connessione CORBA, e così via). Per maggiori informazioni sulla scrittura di codice per eseguire il bind a oggetti del server, consultare la documentazione di VisiBroker.

Utilizzo dell'interfaccia di chiamata dinamica

L'interfaccia di chiamata dinamica (DII) consente alle applicazioni client di chiamare gli oggetti server senza utilizzare una classe stub che coordini in modo esplicito le chiamate dell'interfaccia. Poiché DII deve codificare tutte le informazioni di tipo prima che il client invii una richiesta e quindi deve decodificare tali informazioni sul server, il processo risulta più lento rispetto all'utilizzo di una classe stub.

Per utilizzare la DII in un'applicazione client, si deve

1 Creare un oggetto CORBA generico, che funziona come un oggetto stub, tranne per il fatto che la sua implementazione non include il codice per coordinare le richieste. Sarà necessario fornire l'ID del repository dell'oggetto per indicare quale oggetto dovrebbe ricevere le richieste della DII:

```
CORBA::Object_var diiObj;
try
{
  diiObj = orb->bind("IDL:ServerObj:1.0");
```

```
} catch (const CORBA::Exception& E)
{
  // handle the bind exception.
}
```

2 Quindi, si deve utilizzare l’oggetto CORBA generico per creare un oggetto richiesta per un determinato metodo dell’oggetto rappresentato dall’oggetto CORBA generico:

```
CORBA::Request_var req = diiObj->_request("methodName");
```

3 Quindi, si devono aggiungere tutti gli argomenti accettati dal metodo. Gli argomenti vengono aggiunti a un elenco di richieste di tipo CORBA::NVList_ptr. Ciascuno argomento è di tipo CORBA::Any, che è simile a un Variant. È necessario usare il tipo CORBA::Any poiché un elenco di valori con nome (ORBA::NVList) deve gestire qualsiasi lista di argomenti, che a sua volta può includere valori di qualsiasi tipo.

```
CORBA::Any arg;
arg <<= (const char *)"argvalue";
CORBA::NVList_ptr arguments = req->arguments();
arguments->add_value("arg1", arg, CORBA::ARG_IN);
```

4 A questo punto è possibile inoltrare la richiesta:

```
req->invoke();
```

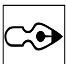
5 nfine, controllare eventuali errori e recuperare il risultato:

```
if (req->env()->exception())
  // handle exception
else
{
  CORBA::Any_ptr pRetVal = req->result()->value();
  CORBA::float val;
  Any_ptr >>= val;
}
```

Per maggiori dettagli sull’utilizzo dell’interfaccia di chiamata dinamica, consultare il manuale *VisiBroker Programmer’s Guide*.

Collaudo di server CORBA

C++Builder include un programma demo, di nome CORBATest, che genera un client universale il quale consente di collaudare le interfacce del server. CORBATest registra e esegue script che sollecitano gli oggetti nell’applicazione server CORBA.



Se si pensa di utilizzare CORBATest per collaudare molti server, potrebbe risultare comodo installare questa applicazione nel menu Tools. Per fare ciò, scegliere Tools | Configure Tools e aggiungere CorbaTest.exe utilizzando la finestra di dialogo Tools Options.

Configurazione dello strumento di collaudo

Prima di poter utilizzare CORBATest per collaudare un'applicazione CORBA, si deve registrare interfaccia per il server con un Interface Repository in esecuzione. Per fare ciò:

1 Assicurarsi che lo SmartAgent di VisiBroker sia attivo scegliendo Tools | VisiBroker SmartAgent nel caso la voce di menu non abbia un segno di spunta.
2 Eseguire l'applicazione server che si vuole collaudare.
3 Scegliere Tools | IDL Repository per aggiungere i file .idl dell'applicazione server all'Interface Repository. Benché sia possibile utilizzare qualsiasi interface repository attivi, probabilmente si vorrà utilizzare un repository dedicato al collaudo in modo che l'operazione non interferisca con altri usi dell'interface repository. Per avviare un repository dedicato al collaudo, fare clic sul pulsante Add IREP nella finestra di dialogo Update IDL Repository, che si richiama scegliendo il comando Tools | IDL Repository.
4 Avviare lo strumento di collaudo mandando in esecuzione CORBATest.exe che è possibile generare a a partire dal codice sorgente del programma dimostrativo presente nella directory corbatest (.. \examples\corba\corbatest).

Registrazione ed esecuzione di script di collaudo

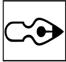
Uno script di collaudo è semplicemente una serie di metodi dell'oggetto CORBA da chiamare e di tutti i parametri che occorre passare a quei metodi. Un singolo script può contenere varie chiamate allo stesso metodo (per esempio, per collaudare il metodo con valori di parametro diversi).

La creazione di uno script avviene aggiungendo comandi al pannello dei comandi nella metà superiore del pannello presente nello strumento di colaudo. Per aggiungere un comando

1 Se l'oggetto, di cui si desidera chiamare un metodo, non ha una pagina dedicata nel pannello degli oggetti (in alto a sinistra), scegliere il comando Edit | Add Object. Apparirà una finestra di dialogo New Object, in cui è possibile selezionare un oggetto in base al suo repository ID. Il repository ID di un oggetto è una stringa in formato "IDL:MyInterface1:1.0", in cui "MyInterface" è il nome dell'interfaccia così come è stata dichiarata nel file .IDL. Quando si seleziona un oggetto, si deve assegnargli un nome.
2 Nella parte in alto a sinistra del pannello, selezionare la pagina dell'oggetto di cui si vuole aggiungere un metodo. Ogni pagina è identificata mediante un nome di oggetto, che è stato assegnato al momento dell'aggiunta dell'oggetto (passo 1). Ogni pagina elenca le operazioni (i metodi) dell'oggetto che è possibile collaudare.
3 Aggiungere i comandi allo script trascinando un'operazione dal pannello oggetti al pannello dei comandi nella metà superiore del pannello dello strumento di collaudo.

4 Quando si seleziona un metodo nel pannello dei comandi, i parametri di input appaiono nel pannello dei dettagli in alto a destra. Immettere i valori per i parametri di input. La barra di stato indica il tipo di valore richiesto per il parametro di input corrente.

Ripetete questi passi per aggiungere tutti i comandi desiderati per tutti gli oggetti desiderati. Non appena lo script contiene tutti i comandi desiderati, scegliere Edit | Save Script As (oppure fare clic sul pulsante Save Script) per salvare lo script e assegnargli un nome. A questo punto è possibile iniziare un nuovo script scegliendo il comando File | New Script (oppure facendo clic sul pulsante New Script).



È possibile rimuovere gli oggetti o i comandi aggiunti a uno script selezionando l'oggetto oppure il comando e scegliendo l'opzione Edit | Remove Object oppure Edit | Remove Command.

È possibile eseguire lo script scegliendo Run | Run, oppure facendo clic sul pulsante Run Script. Durante l'esecuzione dello script, lo strumento di collaudo utilizza l'interfaccia di chiamata dinamica per chiamare il server CORBA, passando i valori di parametro indicati. Il valore di ritorno (se esiste) e tutti i parametri restituiti di tutti i metodi vengono quindi visualizzati nella sezione dei risultati dello strumento di collaudo (oppure in un file di risultati nel caso di collaudo automatico).

La sezione dei risultati visualizza il risultato di tutti i comandi dello script sulla pagina Results. Questi stessi risultati appaiono anche sulle altre pagine della sezioni dei risultati, suddivisi in categorie, in modo da consentire di individuare rapidamente le informazioni desiderate. Le altre pagine della sezione dei risultati includono i valori di ritorno, i valori dei parametri di input, i valori dei parametri di output, e gli eventuali errori verificatisi durante l'esecuzione dello script.

Capitolo

32

Creazione di applicazioni server per Internet

Le applicazioni per server Web estendono le funzioni e le capacità dei server Web esistenti. Un'applicazione per server Web riceve i messaggi di richiesta HTTP dal server Web, compie tutte le azioni richieste in quei messaggi e formula le risposte che restituisce al server Web. Tutte le operazioni che è possibile compiere con un'applicazione C++Builder possono essere incorporata in un'applicazione per server Web.

C++Builder fornisce due architetture diverse per lo sviluppo di applicazioni per Web server: Web Broker e WebSnap. Benché queste due architetture siano diverse, WebSnap e Web Broker hanno molti elementi comuni. L'architettura WebSnap agisce come un superset di Web Broker. Fornisce componenti aggiuntivi e nuove funzioni come Preview tab Designer- che permette di visualizzare il contenuto di una pagina senza che lo sviluppatore debba eseguire l'applicazione. Le applicazioni sviluppate con WebSnap possono includere componenti Web Broker, mentre le applicazioni sviluppate con Web Broker non possono includere componenti WebSnap.

Questo capitolo descrive le caratteristiche delle tecnologie Web Broker e WebSnap e fornisce le informazioni generali sulle applicazioni client/server basate su Internet.

Web Broker e WebSnap

Una delle funzioni di una qualsiasi applicazione è quella di rendere i dati accessibili agli utenti. In un'applicazione standard C++Builder ciò viene fatto creando elementi di interfaccia tradizionali, come finestre di dialogo e finestre con scorrimento. Gli sviluppatori possono specificare la disposizione esatta di questi oggetti utilizzando i soliti strumenti di progettazione schede di C++Builder. Le applicazioni per Web server devono essere però progettate in modo differente. Tutte le informazioni passate agli utenti devono essere sotto forma di pagine HTML trasferite tramite HTTP. Le pagine vengono generalmente interpretate sulla macchina client da

un'applicazione Web browser, che visualizza le pagine in una forma appropriata per il particolare sistema dell'utente nel suo stato attuale.

Il primo passo nella costruzione di un'applicazione per Web server consiste nello scegliere quale architettura si vuole utilizzare, Web Broker o WebSnap. Entrambi gli approcci forniscono molte funzioni simili, tra cui

- Il supporto per applicazioni di tipo CGI e Apache DSO Web server. Questi tipi sono descritti in "Tipi di applicazioni per server Web" a pagina 32-7.
- Supporto per il multithreading per far sì che le richieste in arrivo dei client vengano gestite su thread separati.
- Cache dei moduli Web per risposte più rapide.
- Sviluppo multipiattaforma. L'applicazione per Web server è facilmente portabile tra i sistemi operativi Windows e Linux. Il codice sorgente sarà compilato sull'una o l'altra piattaforma.

Sia i componenti Web Broker che WebSnap gestiscono tutti i meccanismi di trasferimento delle pagine. WebSnap utilizza Web Broker come cardine e pertanto incorpora tutte le funzionalità dell'architettura Web Broker. Tuttavia WebSnap offre un set di strumenti molto più potente per la generazione delle pagine. Inoltre, in fase di esecuzione, le applicazioni WebSnap permettono di utilizzare script lato server come ausilio nella generazione delle pagine. Web Broker non ha questa possibilità di scripting. Gli strumenti offerti in Web Broker non sono affatto completi come quelli in WebSnap e sono molto meno intuitivi. Se si sta sviluppando una nuova applicazione per Web server, probabilmente WebSnap rappresenta una scelta architetturale migliore rispetto a Web Broker.

Le differenze principali fra questi due approcci sono descritte nella seguente tabella:

Tabella 32.1 Web Broker e WebSnap

Web Broker	WebSnap
Compatibile con versioni precedenti	Benché le applicazioni WebSnap possano utilizzare qualsiasi componente Web Broker che produce contenuti, i moduli Web e il dispatcher che li contengono sono nuovi.
In un'applicazione è consentito un solo modulo Web.	Più moduli Web possono suddividere l'applicazione in unità, consentendo a più sviluppatori di lavorare allo stesso progetto con meno conflitti.
Nell'applicazione è consentito un solo dispatcher Web.	Più dispatcher specializzati gestiscono vari tipi di richieste.
I componenti specializzati per la creazione di contenuti includono produttori di pagine, componenti InternetExpress e componenti Web Services.	Supporta tutti i produttori di contenuti che possono apparire in applicazioni Web Broker, oltre a molti altri progettati per consentire la generazione rapida di pagine web complesse data-driven.
Nessun supporto per scripting.	Il supporto per scripting lato server (JScript o VBscript) consente di tenere separata la logica di generazione dell'HTML dalla logica di gestione.

Tabella 32.1 Web Broker e WebSnap

Web Broker	WebSnap
Nessun supporto interno per pagine con nome.	Le pagine con nome possono essere automaticamente ottenute da un dispatcher di pagine e indirizzate da uno script sul lato server.
Nessun supporto per le sessioni.	Le sessioni memorizzano le informazioni necessarie relative a un utente per un breve periodo di tempo. Ciò può essere utilizzato per attività quali il supporto per operazioni di login/logout.
Ogni richiesta deve essere gestita esplicitamente, utilizzando o un elemento di azione o un componente di auto smistamento.	I componenti dispatcher rispondono automaticamente a una vasta gamma di richieste.
Solo alcuni componenti specializzati forniscono un'anteprima del contenuto da essi prodotto. La maggior parte dello sviluppo non è visuale.	WebSnaplets permette di costruire pagine web in modo più visuale e di visualizzare il risultato in fase di progettazione. Le anteprime sono disponibili per tutti i componenti.

Per ulteriori informazioni su Web Broker, vedere il Capitolo 33, "Utilizzo di Web Broker". Per ulteriori informazioni su WebSnap, consultare il Capitolo 34, "Creazione di applicazioni Web Server con WebSnap".

Terminologia e standard

La maggior parte dei protocolli che controllano le attività in Internet è definita nei documenti Request for Comment (RFC) che vengono creati e mantenuti costantemente aggiornati dall'Internet Engineering Task Force (IETF), l'esercito di progetto e sviluppo di protocolli per Internet. Esistono molti RFC importanti risulteranno molto utili durante la scrittura di applicazioni per Internet:

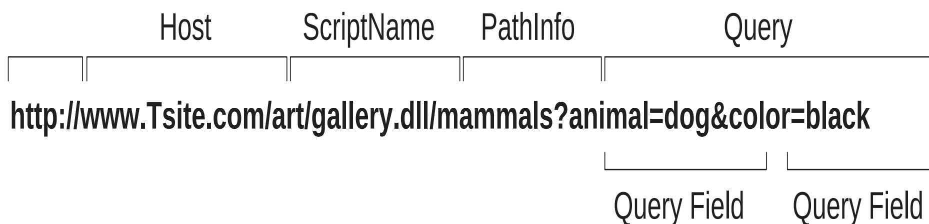
- RFC822, "Standard for the format of ARPA Internet text messages," descrive la struttura e contenuto degli header dei messaggi.
- RFC1521, "MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies," descrive il metodo usato per incapsulare e trasportare messaggi multiparte e multiformato messages.
- RFC1945, "Hypertext Transfer Protocol — HTTP/1.0," descrive un meccanismo di trasferimento usato per distribuire documenti ipermediali in gruppi di lavoro.

Lo IETF conserva una libreria di documenti RFC nel proprio sito Web, `www.ietf.cnri.reston.va.us`

Parti di uno Uniform Resource Locator

Lo Uniform Resource Locator (URL) è una descrizione completa dell'ubicazione di una risorsa disponibile sulla rete. È composto da molte parti a cui può accedere un'applicazione. Queste parti sono illustrate nella Figura 32.1:

Figura 32.1 Parti di uno Uniform Resource Locator



La prima porzione (che tecnicamente non fa parte dell'URL) identifica il protocollo (http). Questa porzione può specificare altri protocolli come https (http sicuro), ftp, e così via.

La porzione Host identifica la macchina su cui è in esecuzione il server web e l'applicazione Web server. Sebbene non sia mostrato nella figura precedente, questa porzione può ridefinire la porta che riceve i messaggi. Di solito, non è necessario specificare una porta, poiché il numero di porta è implicito nel protocollo.

La porzione NomeScript specifica il nome dell'applicazione Web server. Questa è l'applicazione a cui il server web passa i messaggi.

Dopo il nome dell'applicazione è specificata la porzione Pathinfo. Questa porzione identifica la destinazione del messaggio all'interno dell'applicazione Web server. I valori delle informazioni di percorso possono contenere nomi di directory sulla macchina host, nomi di componenti che rispondono a messaggi specifici o qualsiasi altro meccanismo utilizzato dall'applicazione Web server per ripartire l'elaborazione di messaggi in ingresso.

La porzione Query contiene una serie un valori con nome. Questi valori e i rispettivi nomi sono definiti dall'applicazione Web server.

URI e. URL

Lo URL è un sottoinsieme dello Uniform Resource Identifier (URI) definito nel documento relativo agli standard di HTTP, RFC1945. Le applicazioni per server Web spesso producono documenti a partire da varie sorgenti; il risultato finale non risiede in una particolare ubicazione ma è creato all'occorrenza. Gli URI possono descrivere risorse che non hanno una ubicazione specifica.

Informazioni dell'header della richiesta HTTP

I messaggi di richiesta HTTP contengono vari header che descrivono le informazioni sul client, la destinazione della richiesta, il modo in cui dovrebbe essere gestita e gli

eventuali contenuti spediti con la richiesta. Ogni header è identificato da un nome, come, ad esempio, "Host" seguito da un valore di stringa. Per esempio, si esamini la seguente richiesta HTTP:

```
GET /art/gallery.dll/animals?animal=dog&color=black HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.0b4Gold (WinNT; I)
Host: www.TSite.com:1024
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

La prima riga identifica la richiesta come una GET. Un messaggio di richiesta GET chiede all'applicazione per server Web di restituire i contenuti associati allo URI indicato dopo la parola GET (in questo caso /art/gallery.dll/animals?animal=doc&color=black). L'ultima parte della prima riga indica che il client sta utilizzando il protocollo HTTP 1.0 standard.

La seconda riga è lo header della connessione e indica che la connessione non dovrebbe essere chiusa finché la richiesta non è stata assolta. La terza riga è lo header dello User-Agent è fornisce informazioni sul programma che ha generato la richiesta. La riga successiva è lo header dell'Host e fornisce le informazioni sul nome dell'host e sulla porta sul server che è stata contattata per stabilire la connessione. L'ultima riga è lo header di accettazione che elenca i tipi di supporto che il client è in grado di accettare come risposte valide.

Attività di un server HTTP

La natura client/server dei browser per Web è ingannevolmente semplice. Per la maggior parte degli utenti, il recupero di informazioni sul World Wide Web è una procedura semplice: è sufficiente fare clic su un collegamento per far apparire sullo schermo le informazioni. Gli utenti un po' più smaliziati hanno alcune nozioni sulla sintassi HTML e sulla natura client/server dei protocolli usati. Ciò di solito è sufficiente per la produzione di semplici contenuti, orientati alla pagina, per siti Web. Gli autori di pagine Web più complesse hanno un'ampia gamma di opzioni per automatizzare la raccolta e la presentazione di informazioni usando HTML.

Prima di generare un'applicazione per server Web, è utile capire in che modo il client formula una richiesta e il modo in cui il server risponde alle richieste del client.

Composizione delle richieste del client

Quando si seleziona un collegamento ipertestuale HTML (oppure l'utente specifica un URL), il browser raccoglie le informazioni relative al protocollo, al dominio specificato, al percorso per raggiungere le informazioni, sulla data e l'ora, sull'ambiente operativo, sullo stesso browser e su altri contenuti informativi. Quindi compone una richiesta.

Per esempio, per visualizzare una pagina di immagini basata sul criterio selezionato facendo clic sui pulsanti di una scheda, il client costruirebbe questo URL:

```
http://www.TSite.com/art/gallery.dll/animals?animal=dog&color=black
```

che specifica un server HTTP nel dominio www.TSite.com. Il client contatta www.TSite.com, si connette al server HTTP e gli passa una richiesta. La richiesta potrebbe essere simile alla seguente:

```
GET /art/gallery.dll/animals?animal=dog&color=black HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/3.0b4Gold (WinNT; I)
Host: www.TSite.com:1024
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

Soddisfacimento delle richieste del client

Il server Web riceve una richiesta del client e potrebbe eseguire qualsiasi numero di azioni, in base alla sua configurazione. Se il server è configurato per riconoscere come un programma la porzione /gallery.dll della richiesta, esso passa le informazioni sulla richiesta a tale programma. Il modo in cui vengono passate al programma le informazioni sulla richiesta dipendono dal tipo di applicazione per server Web:

- Se il programma è un programma Common Gateway Interface (CGI), il server passa le informazioni contenute nella richiesta direttamente al programma CGI. Il server resta in attesa mentre il programma viene eseguito. Quando il programma CGI termina, esso restituisce i contenuti direttamente al server.
- Se il programma è WinCGI, il server apre un file, vi scrive le informazioni della richiesta. Quindi esegue il programma Win-CGI, passandogli l'ubicazione del file che contiene le informazioni del client e l'ubicazione di un file che il programma Win-CGI dovrebbe usare per creare i contenuti. Il server resta in attesa mentre il programma viene eseguito. Quando il programma termina, il server legge i dati dal file di contenuti scritto dal programma Win-CGI.
- Se il programma è una dynamic-link library (DLL), il server carica la DLL (se occorre) e passa le informazioni contenute nella richiesta alla DLL sotto forma di dato di tipo structure. Il server resta in attesa mentre il programma viene eseguito. Quando la DLL termina, essa restituisce il contenuto direttamente al server.

In tutti i casi, il programma agisce in base alla richiesta e compie le azioni specificate dal programmatore: accede ai database, esegue semplici consultazioni di tabella o calcoli, genera o seleziona documenti HTML e così via.

Risposta alle richieste del client

Dopo aver esaudito una richiesta del client, ogni applicazione per server Web genera una pagina di codice HTML o un altro contenuto MIME e lo restituisce (tramite il server) al client per visualizzarlo. Anche il modo in cui viene trasmessa la risposta è differente, a seconda del tipo di programma:

- Quando un script Win-CGI termina, esso costruisce una pagina HTML, la scrive in un file, scrive tutte le informazioni di risposta in un altro file e restituisce al server le ubicazioni di entrambi i file. Il server apre entrambi i file e restituisce al client la pagina HTML.

- Quando una DLL termina, essa restituisce direttamente la pagina HTML e tutte le informazioni di risposta al server il quale, a sua volta, le restituisce al client.

La creazione di un'applicazione per server Web come DLL riduce il carico di sistema e l'utilizzo delle risorse riducendo il numero dei processi e degli accessi a disco necessari per soddisfare una singola richiesta.

Tipi di applicazioni per server Web

Sia che si utilizzi Web Broker o WebSnap, è possibile creare cinque tipi standard per Web server. Inoltre, è possibile creare un eseguibile Web Application Debugger, che integra il server Web nell'applicazione in modo che sia possibile eseguire il debug della logica dell'applicazione. L'eseguibile Web Application Debugger è destinato solo al debug. Quando si distribuisce l'applicazione, si dovrebbe convertirla in uno degli altri cinque tipi.

ISAPI e NSAPI

Un'applicazione ISAPI o NSAPI per server Web è una DLL che viene caricata dal server Web. Le informazioni della richiesta del client vengono passate alla DLL come una struttura e vengono valutate dall'applicazione ISAPI/NSAPI, che crea la richiesta appropriata e gli oggetti di risposta. Ogni messaggio di richiesta viene gestito automaticamente in un thread di esecuzione separato.

CGI indipendente

Un'applicazione CGI indipendente per server Web è un'applicazione per consolle che riceve le informazioni della richiesta del client sullo standard input e restituisce i risultati al server sullo standard output. Questi dati vengono valutati dall'applicazione CGI, che crea la richiesta appropriata e gli oggetti di risposta. Ogni messaggio di richiesta viene gestito da un'istanza differente dell'applicazione.

Win-CGI indipendente

Un'applicazione Win-CGI indipendente per server Web è un'applicazione Windows che riceve le informazioni della richiesta del client da un file di impostazioni di configurazione (INI) scritto dal server e scrive i risultati in un file che il server restituisce al client. Il file INI viene valutato dall'applicazione del Web server, che crea la richiesta appropriata e gli oggetti di risposta. Ogni messaggio di richiesta viene gestito da un'istanza differente dell'applicazione.

Apache

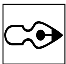
Un'applicazione Apache per server Web è una DLL che viene caricata dal server Web. Le informazioni della richiesta del client vengono passate alla DLL come una struttura e vengono valutate dall'applicazione Apache del Web server, che crea la richiesta appropriata e gli oggetti di risposta. Ogni messaggio di richiesta viene gestito automaticamente in un thread di esecuzione separato.

Quando si distribuisce l'applicazione per Web server Apache, sarà necessario specificare nei file di configurazione di Apache alcune informazioni specifiche per

l'applicazione. Il nome del modulo predefinito è il nome del progetto a cui viene aggiunto il suffisso _module. Ad esempio, il nome del modulo di un progetto di nome Project1 dovrebbe essere Project1_module. Analogamente, il tipo di contenuto predefinito è il nome del progetto a cui viene aggiunto –content, e il tipo di gestore predefinito è il nome del progetto a cui viene aggiunto -handler.

Se occorre, queste definizioni possono essere modificate nel file di progetto (.bpr). Ad esempio, quando si crea il progetto nel file di progetto viene memorizzato un nome di modulo predefinito. Ecco un tipico esempio:

```
extern "C"
{
  Httpd::module __declspec(dllexport) Project1_module;
}
```



Se si cambia nome al progetto durante il processo di salvataggio, quel nome non viene automaticamente modificato. Ogni volta che si rinomina il progetto, è necessario modificare il nome del modulo nel file di progetto in modo che corrisponda al nome del progetto. Le definizioni del contenuto e del gestore dovrebbero cambiare automaticamente una volta che il nome del modulo viene modificato.

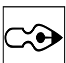
Per informazioni sull'utilizzo nei file di configurazione Apache delle definizioni di modulo, contenuto e gestore, vedere la documentazione sul sito Web di Apache httpd.apache.org.

Web App Debugger

I tipi di server summenzionati presentano vantaggi e svantaggi per gli ambienti di siluppo, ma nessuno di loro è particolarmente adatto per eseguire il debug. La distribuizione dell'applicazione e la configurazione del debugger può rendere il debug dell'applicazione per Web server un'operazione di gran lunga più noiosa rispetto al debug di altri tipi di applicazione.

Per fortuna, il debug di un'applicazione per Web server non è necessariamente così complessa. C++Builder include un Web App Debugger che semplifica le operazioni di debug. Web App Debugger agisce sulla macchina di sviluppo come un server Web. Se si esegue il build dell'applicazione per Web server come un eseguibile di Web App Debugger, la distribuzione viene portata a termine in automatico durante il processo di build. Per eseguire il debug dell'applicazione, è sufficiente avviarla mediante il comando Run|Run. Poi, selezionare Tools|Web App Debugger, fare clic sull'URL predefinito e selezionare l'applicazione nel browser Web che viene visualizzato. L'applicazione verrà avviata nella finestra del browser e sarà possibile utilizzare l'IDE per impostare i breakpoint e ottenere le informazioni di debug.

Non appena l'applicazione sarà pronta per essere collaudata o distribuita in un ambiente di produzione, sarà possibile convertire il progetto Web App Debugger in uno degli altri tipi di destinazione utilizzando la procedura discritta di seguito.



Quando si crea un progetto Web App Debugger, sarà necessario fornire al progetto un CoClass Name. Questo è semplicemente un nome utilizzato da Web App Debugger per fare riferimento all'applicazione. Quasi tutti gli sviluppatori utilizzano come CoClass Name il nome dell'applicazione.

Conversione del tipo di destinazione per applicazioni Web server

Una potente funzionalità di Web Broker e WebSnap è costituita dal fatto che essi offrono vari tipi di server di destinazione. C++Builder consente di eseguire facilmente la conversione da un tipo di destinazione a un altro.

Poiché Web Broker e WebSnap hanno filosofie di progettazione leggermente differenti, è necessario utilizzare un diverso metodo di conversione per ogni architettura. Per convertire il tipo di destinazione dell'applicazione Web Broker, fare quanto segue:

1. Fare clic destro sul modulo Web e scegliere il comando Add To Repository.
2. Nella finestra di dialogo Add To Repository, assegnare al modulo Web un titolo, una descrizione di testo, la pagina del Repository (probabilmente Data Modules), un nome di autore e un'icona.
3. Scegliere OK per salvare il modulo Web come modello.
4. Dal menu principale, scegliere il comando File|New e selezionare l'opzione Web Server Application. Nella finestra di dialogo New Web Server Application scegliere il tipo di destinazione opportuno.
5. Cancellare il modulo Web generato automaticamente.
6. Dal menu principale, scegliere il comando File|New e selezionare il modello salvato al punto 3. Il modello sarà presente nella pagina specificata al punto 2.

Per convertire il tipo di destinazione di un'applicazione WebSnap:

1. Caricare un progetto nell'IDE.
2. Visualizzare Project Manager utilizzando il comando View|Project Manager. Espandere il progetto in modo da rendere visibili tutte le unit.
3. In Project Manager, fare clic sul pulsante New per creare un nuovo progetto di applicazione per Web server. Fare doppio clic nella pagina WebSnap sull'elemento WebSnap Application. Selezionare le opzioni appropriate per il progetto, compreso il tipo server che si desidera utilizzare, quindi fare clic su OK.
4. Espandere il nuovo progetto in Project Manager. Selezionare tutti i file visualizzati e cancellarli.
5. Uno alla volta, selezionare ogni singolo file nel progetto (tranne il file di scheda in progetto Web App Debugger) e trascinarlo sul nuovo progetto. Appena appare una finestra di dialogo che chiede se si desidera aggiungere quel file al nuovo progetto, fare clic su Yes.

Debug di applicazioni server

Il debug di applicazioni per server Web presenta alcuni problemi specifici, perché esse vengono eseguite per rispondere a messaggi provenienti da un server Web. Non è possibile lanciare semplicemente l'applicazione dall'IDE, perché così facendo si

lascia il server Web fuori dal ciclo e l'applicazione non troverà il messaggio di richiesta che si aspettava.

Gli argomenti successivi descrivono le tecniche utilizzabili per eseguire il debug di applicazioni per Web server.

Utilizzo di Web Application Debugger

Web Application Debugger permette di controllare in modo modo semplice le richieste HTTP, le risposte e i tempi di risposta. Il Web Application Debugger sostituisce il server Web. Una volta eseguito il debug dell'applicazione, è possibile convertirla in uno dei tipi supportati di applicazione Web e installarla in un server Web commerciale.

Per utilizzare Web Application Debugger, è necessario per prima cosa creare l'applicazione Web come un eseguibile Web Application Debugger. Sia che si utilizzi Web Broker o WebSnap, il wizard che crea l'applicazione per Web server include questa possibilità come un'opzione da selezionare non appena si inizia l'applicazione. In questo modo viene creata un'applicazione per Web server che è anche un server COM.

Per informazioni su come scrivere questa applicazione per Web server utilizzando Web Broker, vedere il Capitolo 33, "Utilizzo di Web Broker". Per ulteriori informazioni sull'utilizzo di WebSnap, vedere il Capitolo 34, "Creazione di applicazioni Web Server con WebSnap".

Avvio dell'applicazione con Web Application Debugger

Una volta sviluppata l'applicazione per Web server, è possibile eseguirla e farne il debug nel seguente modo:

1. Caricato il progetto nell'IDE, impostare tutti i breakpoint in modo che sia possibile fare il debug dell'applicazione esattamente come con qualsiasi altro eseguibile.
2. Scegliere il comando Run | Run. Viene visualizzata la finestra della console del server COM che è l'applicazione per Web server. La prima volta che si esegue l'applicazione, il server COM viene registrato in modo che il Web Application debugger vi possa accedere.
3. Selezionare il comando Tools | Web App Debugger.
4. Fare clic sul pulsante Start. Viene visualizzata la pagina ServerInfo nel Browser predefinito.
5. La pagina ServerInfo contiene una lista a discesa di tutti gli eseguibili Web Application Debugger registrati. Selezionare l'applicazione dall'elenco a discesa. Se l'applicazione non è presente in questa lista a discesa, provare a eseguire l'applicazione come un normale eseguibile. L'applicazione deve essere eseguita

almeno una volta in modo che possa effettuare la registrazione. Se anche dopo questa operazione l'applicazione non è inclusa nella lista a discesa, provare ad aggiornare la pagina web. (A volte il browser Web memorizza questa pagina nella cache, impedendo di vedere le modifiche più recenti.)

6 Dopo aver selezionato l'applicazione nella lista a discesa, fare clic sul pulsante Go. In questo modo si avvia l'applicazione nel Web Application Debugger, il quale fornisce i dettagli relativi ai messaggi di richiesta e di risposta che passano fra l'applicazione e il Web Application Debugger.

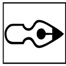
Conversione dell'applicazione in un altro tipo di applicazione per Web server

Terminato il debug dell'applicazione per Web server con Web Application Debugger, sarà necessario convertirla in un altro tipo di applicazione Web che può essere installata su un server Web commerciale. Per maggiori informazioni sulla conversione delle proprie applicazioni, consultare "Conversione del tipo di destinazione per applicazioni Web server" a pagina 32-9.

Debug di applicazioni Web sotto forma di DLL

Le applicazioni ISAPI, NSAPI, e Apache sono in effetti DLL che contengono punti d'ingresso predefiniti. Il server Web passa i messaggi di richiesta all'applicazione facendo chiamate a questi punti d'ingresso. Poiché queste applicazioni sono DLL, è possibile eseguirne il debug impostando i parametri di esecuzione dell'applicazione per avviare il server.

Per configurare i parametri di esecuzione dell'applicazione, scegliere il comando Run | Parameters e impostare Host Application e il Run Parameters in modo da specificare l'eseguibile per il server Web e tutti i parametri necessari nel momento in cui lo si avvia. Per maggiori informazioni sui valori per il proprio server Web, consultare la documentazione fornita dal produttore del Web Server.



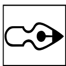
Per alcuni Web Servers è necessario apportare ulteriori modifiche prima che si abbia il diritto di avviare l'Host Application in questo modo. Per i dettagli, consultare la documentazione fornita dal produttore del Web Server.



Se si utilizza Windows 2000 con IIS 5, i dettagli su tutte le modifiche che è necessario apportare per configurare correttamente i diritti sono descritte sul sito Web seguente:

`http://community.borland.com/article/0,1410,23024,00.html`

Una volta configurate le opzioni Host Application e Run Parameters, è possibile configurare i breakpoint in modo che, non appena il server passa un messaggio di richiesta alla DLL, uno di questi breakpoint venga intercettato e si possa eseguire il debug normalmente.



Prima di avviare il server Web usando i parametri di esecuzione dell'applicazione, assicurarsi che il server non sia già in esecuzione.

Diritti utente necessari per il debug di DLL

In ambiente Windows, si devono avere i diritti di utenza adeguati per eseguire il debug di una DLL. È possibile ottenere i diritti nel seguente modo:

1 Nella parte Administrative Tools del Control Panel, fare clic su Local Security Policy. Espandere Local Policies e fare doppio clic su User Rights Assignment. Fare doppio clic sull'opzione Act as part of the operating system nella parte destra del pannello.

2 Selezionare Add per aggiungere un utente all'elenco. Aggiungere l'utente corrente.

3 Riavviare il sistema in modo da rendere operative le modifiche.

Capitolo

33

Utilizzo di Web Broker

I componenti di Web Broker (situati sulla pagina Internet della Component palette) permettono di creare gestori di evento che sono associati a uno specifico Uniform Resource Identifier (URI). A elaborazione conclusa, è possibile impostare da programma documenti HTML o XML e trasferirli al client. I componenti WEb Broker si possono utilizzare per lo sviluppo di applicazioni per più piattaforme.

Spesso, il contenuto delle pagine web è tratto da dei database. È possibile utilizzare dei componenti Internet per gestire automaticamente le connessioni ai database, il che consente a un singolo DLL di gestire numerose connessioni database thread-safe.

Nelle prossime sezioni di questo capitolo viene spiegato come si usano i componenti Web Broker per creare un'applicazione per Web server.

Creazione di applicazioni per Web server con Web Broker

Per creare una nuova applicazione per Web server utilizzando l'architettura Web Broker:

1 Selezionare File | New | Other.

2 Nella finestra di dialogo New Items, selezionare la pagina New e scegliere Web Server Application.

3 Viene visualizzata una finestra di dialogo, dove è possibile selezionare uno dei tipi di applicazione Web server:

- ISAPI and NSAPI: Selezionando questo tipo di applicazione si configura il progetto come un DLL, con i metodi esportati previsti dal Web server. Include inoltre i file header appropriati per la generazione dell'applicazione.
- Apache: Selezionando questo tipo di applicazione si configura il progetto come un DLL, con i metodi esportati previsti dal Web server Apache. Include inoltre i file header appropriati per la generazione dell'applicazione.

- CGI stand-alone: La selezione di questo tipo di applicazione imposta il progetto come un'applicazione per console e include i file header appropriati.
- Win-CGI stand-alone: La selezione di questo tipo di applicazione imposta il progetto come un'applicazione per Windows e include i file header appropriati.
- Web Application Debugger stand-alone executable: La selezione di questo tipo di applicazione configura un ambiente per sviluppare e provare applicazioni per Web server. Questo tipo di applicazione non è destinato all'impiego.

Scegliere il tipo di Web *Server Application* capace di comunicare con il tipo di Web Server che l'applicazione utilizzerà. In questo modo, si crea un nuovo progetto configurato per utilizzare i componenti Internet e contenente un modulo Web vuoto.

Il modulo Web

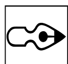
Il modulo Web (*TWebModule*) è un discendente di *TDataModule* e può essere utilizzato nello stesso modo: fornire il controllo centralizzato delle regole di gestione e dei componenti non visuali nell'applicazione Web.

Si aggiunge poi qualsiasi produttore di contenuti che l'applicazione utilizza per generare messaggi di risposta. Questi possono essere i produttori di contenuti incorporati, come *TPageProducer*, *TDataSetPageProducer*, *TDataSetTableProducer*, *TQueryTableProducer* e *TInetXPageProducer* oppure discendenti realizzati ad hoc di *TCustomContentProducer*. Se l'applicazione genera messaggi di risposta comprendenti materiale tratto da database, si possono aggiungere componenti di accesso ai dati o componenti speciali per scrivere un Web server che si comporti da cliente in un'applicazione per database multi-tier.

Oltre registrare i componenti non visuali e le regole di gestione, il modulo Web agisce anche come dispatcher, e fa corrispondere i messaggi di richiesta HTTP in arrivo a elementi di azione che generano le risposte alle richieste.

A volte, si dispone già di un modulo dati contenente molti dei componenti e delle regole di gestione non visuali che si intendono usare nell'applicazione Web. È possibile sostituire il modulo Web con il proprio modulo dati preesistente. È sufficiente cancellare il modulo Web generato automaticamente e sostituirlo con il proprio modulo dati. Quindi, si aggiunge al modulo dati un componente *TWebDispatcher*, in modo che possa eseguire il dispatch di messaggi di richiesta a elementi di azione, secondo le modalità previste per i moduli Web. Se si desidera modificare il modo in cui gli elementi di azione sono selezionati per rispondere a messaggi di richiesta HTTP in arrivo, far derivare un nuovo componente dispatcher da *TCustomWebDispatcher* e aggiungerlo al modulo dati al posto dell'altro.

Il progetto può contenere solamente un dispatcher. Questo può essere sia il modulo Web generato automaticamente quando si crea il progetto, sia il componente *TWebDispatcher* aggiunto a un modulo dati in sostituzione del modulo Web. Se in fase di l'esecuzione si crea un secondo modulo dati contenente un dispatcher, l'applicazione per Web server genera un errore di esecuzione.



Il modulo Web impostato in fase di progetto è in effetti un modello. Nelle applicazioni ISAPI e NSAPI, ogni messaggio di richiesta genera un thread distinto e

per ogni thread vengono create dinamicamente ricorrenze distinte del modulo Web e del suo contenuto.



Il modulo Web di un'applicazione per Web server basata su DLL viene messo in cache per migliorare il tempo di risposta quando viene riutilizzato. Lo stato del dispatcher e della sua lista di azioni non viene reinizializzato tra due richieste successive. L'attivazione o la disattivazione di elementi di azione in fase di esecuzione può produrre risultati inattesi, quando il modulo è utilizzato per successive richieste client.

L'oggetto Application Web

Il progetto configurato per l'applicazione Web contiene una variabile globale chiamata *Application*. *Application* è un discendente di *TWebApplication* (*TISAPIApplication*,*TApacheApplication* o *TCGIApplication*) adeguato al tipo di applicazione che si sta creando. Viene eseguito in risposta a messaggi di richiesta HTTP provenienti dal Web server.



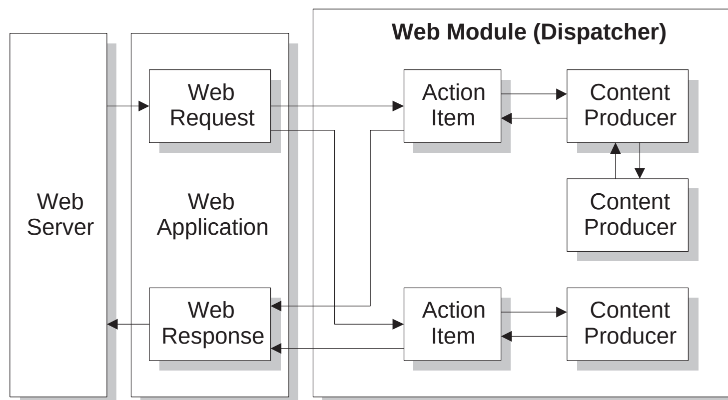
Nel file di progetto non includere Forms.hpp dopo l'istruzione include relativa a CGIApp.hpp o a ISAPIApp.hpp. Forms.hpp dichiara tra l'altro una variabile globale di nome *Application*, e se tale variabile appare dopo CGIApp.hpp o ISAPIApp.hpp, *Application* verrà inizializzata a un oggetto di tipo errato.

Struttura delle applicazioni Web Broker

Quando l'applicazione Web riceve un messaggio di richiesta HTTP, crea un oggetto *TWebRequest* che rappresenta il messaggio di richiesta HTTP e un oggetto *TWebResponse* per rappresentare la risposta da restituire. L'applicazione quindi passa questi oggetti al dispatcher Web (che può essere il modulo Web o un componente *TWebDispatcher*).

Il dispatcher Web controlla il flusso dell'applicazione per Web server. Il dispatcher gestisce un assortimento di elementi di azione (*TWebActionItem*) che sanno come gestire certi tipi di messaggi di richiesta HTTP. Il dispatcher identifica gli elementi di azione appropriati o i componenti auto-dispatching capaci di gestire il messaggio di richiesta HTTP e passa gli oggetti di richiesta e di risposta al gestore identificato per consentirgli di eseguire l'azione richiesta o di formulare un messaggio di risposta. Tutto ciò viene spiegato in dettaglio nella sezione "Il dispatcher Web" a pagina 33-4.

Figura 33.1 Struttura di un'applicazione server



Gli elementi di azione hanno la responsabilità di leggere la richiesta e di mettere insieme il messaggio di risposta. I produttori di contenuti specializzati affiancano gli elementi di azione nel generare dinamicamente il contenuto dei messaggi di risposta, che possono comprendere un codice HTML ad hoc, o altro contenuto MIME. I produttori di contenuti possono fare uso di altri produttori o discendenti di *THTMLTagAttributes*, per aiutarli a compilare il messaggio di risposta. Per maggiori informazioni sui produttori di contenuti, consultare "Generazione del contenuto dei messaggi di risposta" on pagina 33-14.

Se si crea il Web Client in un'applicazione per database multi-tier, l'applicazione per Web server può includere altri componenti auto-dispatching che rappresentano informazioni di database codificate in XML e classi di manipolazione di database codificate in javascript. Se si crea un server che implementa un Web Service, l'applicazione per Web server può includere un componente auto-dispatching che passa messaggi basati su SOAP a un invoker che li interpreta e li esegue. Il dispatcher invita i componenti auto-dispatching a gestire il messaggio di richiesta, dopo averci provato con tutti i suoi elementi di azione.

Quando tutti gli elementi di azione (o i componenti per il dispatch automatico) hanno terminato di creare la risposta riempiendo l'oggetto *TWebResponse*, il dispatcher restituisce il risultato all'applicazione Web. L'applicazione trasmette la risposta al client tramite il server Web.

Il dispatcher Web

Se si utilizza un modulo Web, questo agisce come un dispatcher Web. Se si utilizza un modulo dati esistente, bisogna aggiungervi un singolo componente dispatcher (*TWebDispatcher*). Il dispatcher gestisce un assortimento di elementi di azione in grado di gestire certi tipi di messaggi di richiesta. Quando l'applicazione Web passa un oggetto di richiesta e un oggetto di risposta al dispatcher, seleziona anche uno o più elementi di azione per rispondere alla richiesta.

Aggiunta di azioni al dispatcher

Si apre l'action editor di Object Inspector facendo clic sui puntini di sospensione della proprietà *Actions* del dispatcher. Gli elementi di azione possono essere aggiunti al dispatcher facendo clic sul pulsante Add nell'Action editor.

Si aggiungono azioni al dispatcher per rispondere a vari metodi di richiesta o agli URI di destinazione. Vi sono diversi modi di impostare gli elementi di azione. Si può cominciare da quelli che preelaborano le richieste e terminano con un'azione predefinita che verifica se la risposta è completa e invia la risposta, oppure restituisce un codice di errore. Si può anche aggiungere un elemento di azione distinto per ogni tipo di richiesta. In questo caso, ciascun elemento di azione gestisce indipendentemente la propria richiesta.

Gli elementi di azione sono discussi dettagliatamente in "Elementi di azione" on pagina 33-6.

Dispatch dei messaggi di richiesta

Quando il dispatcher riceve la richiesta client, genera un evento *BeforeDispatch*. Questo dà all'applicazione la possibilità di preelaborare il messaggio di richiesta prima che sia visto da un elemento di azione.

Quindi, il dispatcher cerca nell'elenco degli elementi di azione un elemento che coincida con la porzione *PathInfo* dell'*URL* di destinazione del messaggio di richiesta e che possa fornire il servizio specificato come metodo del messaggio di richiesta. Allo scopo, confronta le proprietà *PathInfo* e *MethodType* dell'oggetto *TWebRequest* con le proprietà dello stesso nome presenti nell'elemento di azione.

Quando il dispatcher trova un elemento di azione appropriato, lo attiva. L'elemento di azione una volta attivato, fa una delle cose seguenti:

- Compila il contenuto della risposta e lo invia, oppure segnala che la richiesta è completamente gestita.
- Aggiunge qualcosa alla risposta e poi permette ad altri elementi di azione di completare il lavoro.
- Passa la richiesta ad altri elementi di azione.

Se, dopo avere controllato tutti i suoi elementi di azione, il messaggio non è stato gestito, il dispatcher controlla ogni componente auto-dispatching registrato in modo speciale che non utilizza elementi di azione. Questi componenti sono specifici per le applicazioni database multi-tiered e sono descritti nella sezione "Creazione di applicazioni Web con InternetExpress" a pagina 29-33

Se, dopo avere controllato tutti gli elementi di azione e qualsiasi componente auto-dispatching registrato in modo speciale, il messaggio di richiesta non è stato ancora completamente gestito, il dispatcher chiama l'elemento di azione predefinito. L'elemento di azione predefinito non ha bisogno di corrispondere né all'*URL* di destinazione né al metodo della richiesta.

Quando il dispatcher ha raggiunto la fine della lista di azioni (tra cui l'azione predefinita, se prevista) e nessuna azione è stata attivata, al server non viene inviato nulla. Il server si limita a interrompere la connessione con il client.

Se la richiesta è gestita dagli elementi di azione, il dispatcher genera un evento *AfterDispatch*. Questo offre all'applicazione l'ultima opportunità di verificare la risposta fornita e di applicare modifiche all'ultimo minuto.

Elementi di azione

Ogni elemento di azione (*TWebActionItem*) esegue un'operazione specifica in risposta a un certo tipo di messaggio di richiesta.

Gli elementi di azione possono rispondere completamente a una richiesta, o eseguire parte della risposta lasciando poi che siano altri elementi di azione a completare il lavoro. Gli elementi di azione possono inviare il messaggio HTTP di risposta alla richiesta, o semplicemente impostare una parte della risposta che viene poi completata da altri elementi di azione. Se una risposta è completata dagli elementi di azione ma non viene inviata, l'applicazione per Web server invia il messaggio di risposta.

Determinazione dell'attivazione degli elementi di azione

La maggior parte delle proprietà di un elemento di azione determinano quando il dispatcher lo seleziona per gestire un messaggio di richiesta HTTP. Per impostare le proprietà di un elemento di azione, è necessario attivare prima l'Action editor: si seleziona la proprietà *Actions* del dispatcher nell'*Object* Inspector e si fa clic sui puntini di sospensione. Quando un'azione è selezionata nell'Action editor, se ne possono modificare le proprietà nell'*Object* Inspector.

L'URL di destinazione

Il dispatcher confronta la proprietà *PathInfo* di un elemento di azione con il *PathInfo* del messaggio di richiesta. Il valore di questa proprietà dovrebbe essere la parte riguardante le informazioni di percorso dell'URL per tutte le richieste che l'elemento di azione è preparato a gestire. Ad esempio, dato questo URL:

```
http://www.TSite.com/art/gallery.dll/mammals?animal=dog&color=black
```

e supponendo che la parte `/gallery.dll` indichi l'applicazione per Web server, la parte riguardante le informazioni di percorso è

```
/mammals
```

Le informazioni di percorso sono utili per indicare dove l'applicazione Web deve cercare le informazioni quando esegue le richieste o per dividere l'applicazione Web in sottoinsiemi logici.

La proprietà Method Type

La proprietà *MethodType* di un elemento di azione indica quale tipo di messaggi di richiesta può elaborare. Il dispatcher confronta la proprietà *MethodType* di un elemento di azione al *MethodType* del messaggio di richiesta. *MethodType* può assumere uno dei seguenti valori:

Tabella 33.1 Valori di MethodType

Valore	Significato
mtGet	La richiesta chiede alle informazioni associate all'URI di destinazione di essere restituita in un messaggio di risposta.
mtHead	La richiesta richiede le proprietà dell'intestazione di una risposta, come se rispondesse a una richiesta *mtGet*, ma omettendo il contenuto della risposta.
mtPost	La richiesta fornisce informazioni da inviare all'applicazione Web.
mtPut	La richiesta chiede che la risorsa associata all'URI di destinazione sia sostituita dal contenuto del messaggio di richiesta.
mtAny	Corrisponde a qualsiasi tipo di metodo di richiesta, tra cui *mtGet*, *mtHead*, *mtPut* e *mtPost*.

Attivazione e disattivazione di elementi di azione

Ogni elemento di azione ha una proprietà *Enabled* che può essere utilizzata per attivarlo o disattivarlo. Impostando *Enabled* a **false**, si disattiva l'elemento di azione facendo sì che esso non venga preso in considerazione dal dispatcher quando ricerca un elemento di azione per gestire una richiesta.

Un gestore di evento *BeforeDispatch* può controllare quali elementi di azione devono elaborare una richiesta modificando la loro proprietà *Enabled*, prima che il dispatcher cominci a farli corrispondere al messaggio di richiesta.



Modificare la proprietà *Enabled* durante l'esecuzione di un'azione può determinare risultati inattesi nelle richieste successive. Se l'applicazione per Web server è un DLL che mette in cache i moduli Web, alla richiesta successiva non verrà ripristinato lo stato iniziale. Per riportare tutti gli elementi di azione allo stato iniziale opportuno si utilizza l'evento *BeforeDispatch*.

Scelta di un elemento di azione predefinito

Solo uno degli elementi di azione può essere l'elemento di azione predefinito. L'elemento di azione predefinito viene selezionato impostandone la proprietà *Default* a **true**. Quando la proprietà *Default* di un elemento di azione viene impostata a **true**, la proprietà *Default* del precedente elemento di azione predefinito (se previsto) viene impostata a **false**.

Quando il dispatcher esamina la lista di elementi di azione per sceglierne uno che sappia gestire una richiesta, memorizza il nome dell'elemento di azione predefinito. Se la richiesta non è stata gestita completamente quando il dispatcher ha raggiunto la fine della lista di elementi di azione, viene eseguito l'elemento di azione predefinito.

Il dispatcher non controlla le proprietà *PathInfo* o *MethodType* dell'elemento di azione predefinito. Il dispatcher non controlla neanche la proprietà *Enabled*. Pertanto, si può

essere certi che l'elemento di azione predefinito venga chiamato solamente alla fine, impostandone la proprietà *Enabled* a **false**.

L'elemento di azione predefinito dovrebbe essere preparato a gestire qualsiasi richiesta che incontra, anche se è solo per restituire un codice di errore indicante una URI o un *MethodType* non validi. Se l'elemento di azione predefinito non gestisce la richiesta, nessuna risposta è inviata al client Web.



Modificare la proprietà *Default* di un'azione durante l'esecuzione potrebbe produrre risultati inattesi per la richiesta corrente. Se la proprietà *Default* di un'azione già attivata è impostata a **true**, l'azione non viene riesaminata e il dispatcher non la attiva nuovamente quando raggiunge la fine della lista di azioni.

Risposta a messaggi di richiesta mediante elementi di azione

Il lavoro effettivo di un'applicazione per Web server viene svolto dagli elementi di azione quando vengono eseguiti. Quando il dispatcher Web attiva un elemento di azione, questo può reagire in due modi al messaggio di richiesta corrente:

- Se all'elemento di azione è associato come valore della sua proprietà *Producer* un componente produttore, questo assegna automaticamente il *Content* del messaggio di risposta utilizzando il proprio metodo *Content*. La pagina Internet della Component palette contiene molti componenti produttori di contenuti utili a costruire una pagina HTML contenente il messaggio di risposta.
- Dopo che il produttore ha assegnato il contenuto della risposta (se c'è un produttore associato), l'elemento di azione riceve un evento *OnAction*. Al gestore di evento *OnAction* vengono passati l'oggetto *TWebRequest* rappresentante il messaggio di richiesta HTTP e un oggetto *TWebResponse* per compilare le informazioni di risposta.

Se il contenuto dell'elemento di azione può essere generato da un unico produttore di contenuti, è più semplice assegnarlo come proprietà *Producer* dell'elemento di azione.Tuttavia, mediante il gestore di evento *OnAction* si può sempre accedere anche a qualsiasi produttore di contenuti. Il gestore di evento *OnAction* consente più flessibilità , in modo che sia possibile utilizzare più produttori di contenuti, stabilire le proprietà dei messaggi di risposta, e così via.

Per rispondere ai messaggi di richiesta sia il componente produttore di contenuti che il gestore di evento *OnAction* possono utilizzare qualsiasi oggetto e qualsiasi metodo di libreria. Infatti, possono accedere a database, eseguire calcoli, comporre o selezionare documenti HTML e così via. Per maggiori informazioni sulla generazione di contenuti di risposta utilizzando componenti produttori di contenuti, consultare la sezione "Generazione del contenuto dei messaggi di risposta" a pagina 33-14.

Invio della risposta

Un gestore di evento *OnAction* può spedire la risposta al client Web utilizzando i metodi dell'oggetto *TWebResponse*. Tuttavia, se nessun elemento di azione invia la risposta al client, questa verrà spedita dall'applicazione per Web server fino a quando l'ultimo elemento di azione che esaminerà la richiesta non indicherà che essa è stata portata a termine.

Utilizzo di più elementi di azione

È possibile rispondere a una richiesta usando un singolo elemento di azione, oppure ripartendo il lavoro tra più elementi. Se l'elemento di azione non termina completamente la preparazione del messaggio di risposta, deve segnalare questo stato nel gestore di evento *OnAction* mediante l'impostazione a **false** del parametro *Handled*.

Se molti elementi di azione si suddividono il compito di rispondere ai messaggi di richiesta, ciascuno impostando *Handled* a **false** in modo da consentire agli altri di continuare, assicurarsi che l'elemento di azione predefinito lasci il parametro *Handled* impostato a **true**. Altrimenti, al client Web non verrà inviata nessuna risposta.

Quando si ripartisce il lavoro tra più elementi di azione, il gestore di evento *OnAction* dell'elemento di azione predefinito oppure il gestore di evento *AfterDispatch* del dispatcher devono verificare se l'incombenza è stata portata a termine e riportare il codice di errore appropriato in caso contrario.

Accesso alle informazioni della richiesta client

Quando l'applicazione per Web server riceve un messaggio di richiesta HTTP, l'intestazione della richiesta client viene caricata nelle proprietà di un oggetto *TWebRequest*. Nelle applicazioni NSAPI e ISAPI, il messaggio di richiesta è incorporato in un oggetto *TISAPIRequest*. Le applicazioni CGI di console utilizzano oggetti *TCGIRequest*, mentre le applicazioni Windows CGI utilizzano oggetti *TWinCGIRequest*.

Le proprietà dell'oggetto di richiesta sono di sola lettura. Queste possono essere utilizzate per raccogliere tutte delle informazioni disponibile nella richiesta client.

Proprietà contenenti informazioni dello header della richiesta

In un oggetto di richiesta, la maggior parte delle proprietà contengono informazioni riguardanti la richiesta indicate nello header della richiesta HTTP. Non tutte le richieste forniscono il valore di ognuna di queste proprietà. Inoltre, alcune richieste possono comprendere campi di intestazione che non affiorano in una proprietà dell'oggetto di richiesta, soprattutto perché lo standard HTTP è in costante evoluzione. Per ottenere il valore di un campo dello header della richiesta non identificato come proprietà dell'oggetto richiesta, si ricorre al metodo *GetFieldByName*.

Proprietà che identificano la destinazione

La destinazione completa di un messaggio di richiesta è data dalla sua proprietà *URL*. Di solito, è una URL che si può suddividere in protocollo (HTTP), *Host* (sistema server), *ScriptName* (applicazione server), *PathInfo* (ubicazione sull'host) e *Query*.

Ogni segmento emerge nella propria proprietà. Il protocollo è sempre HTTP, mentre *Host* e *ScriptName* identificano l'applicazione per Web server. Il dispatcher utilizza la parte *PathInfo* quando fa corrispondere elementi di azione ai messaggi di richiesta.

Query è utilizzato da alcune richieste per specificare i dettagli delle informazioni chieste. Il suo valore viene analizzato come la proprietà *QueryFields*.

Proprietà che descrivono il client Web

La richiesta include anche molte proprietà che forniscono informazioni sull'origine della richiesta. Queste includono tutte le informazioni disponibili, dall'indirizzo di posta elettronica del mittente (proprietà *From*) all'URI da cui ha avuto origine il messaggio (proprietà *Referer* o *RemoteHost*). Se la richiesta include un contenuto non proveniente dalla URI da cui proviene la richiesta, l'origine del contenuto è data dalla proprietà *DerivedFrom*. Si possono anche determinare l'indirizzo IP del client (la proprietà *RemoteAddr*) e il nome e la versione dell'applicazione che ha inviato la richiesta (la proprietà *UserAgent*).

Proprietà che identificano lo scopo della richiesta

La proprietà *Method* è una stringa che descrive quello che il messaggio di richiesta chiede di fare all'applicazione server. Lo standard HTTP 1.1 definisce i seguenti metodi:

Valore	Che cosa richiede il messaggio
OPTIONS	Informazioni sulle opzioni di comunicazione disponibili.
GET	Informazioni identificate dalla proprietà *URL*.
HEAD	Informazioni di intestazione da un messaggio equivalente *GET*, senza il contenuto della risposta.
POST	All'applicazione server di registra i dati inclusi nella proprietà *Content*, come appropriato.
PUT	All'applicazione server di sostituire nella risorsa indicata dalla proprietà *URL* i dati presenti nella proprietà *Content*.
DELETE	All'applicazione server di cancellare o nascondere la risorsa identificata dalla proprietà *URL*.
TRACE	All'applicazione server di inviare una "ricevuta di ritorno" per confermare di aver ricevuto la richiesta.

La proprietà *Method* può indicare qualsiasi altro metodo che il client Web richiede al server.

L'applicazione per Web server non deve fornire una risposta a ogni possibile valore di *Method*. Lo standard HTTP vuole però che esso tratti sia le richieste *GET* che quelle *HEAD*.

La proprietà *MethodType* indica se il valore di *Method* è *GET* (*mtGet*), *HEAD* (*mtHead*), *POST* (*mtPost*), *PUT* (*mtPut*) o un'altra stringa (mtAny). Il dispatcher fa corrispondere il valore della proprietà *MethodType* al *MethodType* di ogni elemento di azione.

Proprietà che descrivono la risposta prevista

La proprietà *Accept* indica i tipi di supporto mediatico che il client Web accetta come contenuto del messaggio di risposta. La proprietà *IfModifiedSince* specifica se il client vuole solo informazioni che sono cambiate recentemente. La proprietà *Cookie* include

informazioni di stato (di solito aggiunte precedentemente dall'applicazione) che possono modificare la risposta.

Proprietà che descrivono il contenuto

La maggior parte delle richieste sono prive di contenuto, in quanto chiedono solo informazioni. Tuttavia, alcune richieste, come le richieste *POST*, forniscono un contenuto che si prevede verrà utilizzato dall'applicazione per Web server. Il tipo di supporto del contenuto è fornito nella proprietà *ContentType* e la sua lunghezza nella proprietà *ContentLength*. Se il contenuto del messaggio è stato codificato (ad esempio, per la compressione dati), queste informazioni si trovano nella proprietà *ContentEncoding*. Il nome e il numero di versione dell'applicazione che ha prodotto il contenuto è specificato dalla proprietà *ContentVersion*. Anche la proprietà *Title* può fornire informazioni sul contenuto.

Il contenuto dei messaggi di richiesta HTTP

Oltre ai campi di intestazione, alcuni messaggi di richiesta includono una porzione di contenuto che l'applicazione per Web server deve elaborare in qualche modo. Ad esempio, una richiesta *POST* può includere informazioni da aggiungere a un database gestito dall'applicazione per Web server.

Il valore non elaborato del contenuto è dato dalla proprietà *Content*. Se questo può essere suddiviso in campi separati da simboli (&), una versione scomposta è disponibile nella proprietà *ContentFields*.

Creazione di messaggi di risposta HTTP

Quando l'applicazione per Web server crea un oggetto *TWebRequest* in risposta a un messaggio di richiesta HTTP in arrivo, crea anche un oggetto corrispondente *TWebResponse* che rappresenta il messaggio di risposta che sarà inviato. Nelle applicazioni NSAPI e ISAPI, il messaggio di risposta è incorporato in un oggetto *TISAPIResponse*. Le applicazioni CGI di console utilizzano oggetti *TCGIResponse*, mentre le applicazioni Windows CGI utilizzano oggetti *TWinCGIResponse*.

Gli elementi di azione che generano la risposta a una richiesta Web client riempiono le proprietà dell'oggetto di risposta. In alcuni casi, è una cosa semplice, come restituire un codice di errore, o inoltrare la richiesta a un altro URI. In altri casi, possono essere necessari calcoli complessi che richiedono all'elemento di azione di prelevare informazioni da altre sorgenti e di dare loro una forma finita. La maggior parte dei messaggi di richiesta richiedono una risposta, anche se se si tratta solo della conferma che l'azione richiesta è stata eseguita.

Compilazione dell'intestazione della risposta

La maggior parte delle proprietà dell'oggetto *TWebResponse* rappresentano le informazioni di intestazione del messaggio di risposta HTTP inviato al client Web.

Un elemento di azione imposta queste proprietà prendendole dal proprio gestore di evento *OnAction*.

Non tutti i messaggi di risposta devono specificare il valore di ciascuna proprietà dell'intestazione. Le proprietà da impostare dipendono dalla natura della richiesta e dallo stato della risposta.

Indicazione dello stato della risposta

Ogni messaggio di risposta deve includere un codice di stato che indica lo stato della risposta. Il codice di stato si imposta mediante la proprietà *StatusCode*. Lo standard HTTP definisce molti codici di stato standard dal significato predefinito. Inoltre, si possono definire codici di stato non standard utilizzando qualsiasi valore inutilizzato ammissibile.

Ogni codice di stato è un numero di tre cifre dove la cifra più significativa indica la classe della risposta, come segue:

- 1xx: Informational (la richiesta è stata ricevuta ma non è stata elaborata completamente).
- 2xx: Success (la richiesta è stata ricevuta, capita e accettata).
- 3xx: Redirection (un'ulteriore iniziativa da parte del client è necessaria per completare la richiesta).
- 4xx: client Error (la richiesta non è comprensibile o non può essere eseguita).
- 5xx: Server Error (la richiesta era valida ma il server non ha potuto gestirla).

A ogni codice di stato è associata una stringa che ne spiega il significato. Si tratta della proprietà *ReasonString*. Nel caso dei codici di stato predefiniti, non è necessario impostare la proprietà *ReasonString*. Se invece si definiscono codici di stato non standard, è bene impostare la proprietà *ReasonString*.

Indicazione della necessità di azione da parte del client

Quando il codice di stato è compreso nell'intervallo 300-399, il client deve eseguire un'ulteriore azione prima che l'applicazione per Web server possa portare a termine la sua richiesta. Se è necessario dirigere il client verso un altro URI, o indicare che per gestire la richiesta è stato creato un nuovo URI, si deve impostare la proprietà *Location*. Se il client deve fornire una password prima che sia possibile procedere, si imposta la proprietà *WWWAuthenticate*.

Descrizione dell'applicazione server

Alcune delle proprietà di intestazione della risposta descrivono le possibilità dell'applicazione per Web server.La proprietà *Allow* indica i metodi a cui l'applicazione può rispondere. La proprietà *Server* dà il nome e il numero di versione dell'applicazione utilizzata per generare la risposta. La proprietà *Cookies* può ospitare informazioni di stato sull'utilizzo da parte del client dell'applicazione server contenuta nei successivi messaggi di richiesta.

Descrizione del contenuto

Numerose proprietà descrivono il contenuto della risposta. *ContentType* indica il tipo di supporto mediatico della risposta e *ContentVersion* è il numero di versione relativo a quel tipo di supporto. *ContentLength* dà la lunghezza della risposta. Se il contenuto è codificato (come nel caso di una compressione dati), lo si segnala tramite la proprietà *ContentEncoding*. Se il contenuto proviene da un altro URI, lo si indica nella proprietà *DerivedFrom*. Se il valore del contenuto dipende dal tempo, la proprietà *LastModified* e la proprietà *Expires* indicano se il valore è ancora valido. La proprietà *Title* fornisce informazioni descrittive sul contenuto.

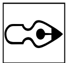
Impostazione del contenuto della risposta

Per alcune richieste, la risposta al messaggio di richiesta è contenuta completamente nelle proprietà dell'intestazione della risposta. Comunque, nella maggior parte dei casi, l'elemento di azione assegna parte dei contenuti al messaggio di risposta. Tali contenuti possono essere informazioni statiche registrate in un file o informazioni prodotte dinamicamente dall'elemento di azione o dal proprio produttore di contenuti.

È possibile impostare il contenuto del messaggio di risposta utilizzando la proprietà *Content* oppure la proprietà *ContentStream*.

La proprietà *Content* è una AnsiString. Le AnsiString di C++Builder non sono limitate a valori del testo, e pertanto il valore della proprietà *Content* può essere una stringa di comandi HTML, oppure contenuti grafici sotto forma di stream di bit, o qualsiasi altro contenuto di tipo MIME.

Si utilizza la proprietà *ContentStream* se il contenuto del messaggio di risposta può essere letto da un flusso. Ad esempio, se il messaggio di risposta deve inviare il contenuto di un file, si utilizza un oggetto *TFileStream* per la proprietà *ContentStream*. Come nel caso della proprietà *Content*, *ContentStream* può fornire una stringa di comandi HTML o un altro tipo di contenuto MIME. Quando viene usata la proprietà *ContentStream*, non c'è bisogno di liberare esplicitamente il flusso. E' l'oggetto di risposta Web che lo libera automaticamente.



Se il valore della proprietà *ContentStream* non è NULL, la proprietà *Content* viene ignorata.

Invio della risposta

Se si ha la certezza che non c'è altro lavoro da fare per rispondere a un messaggio di richiesta, si può far spedire la risposta direttamente da un gestore di evento *OnAction*. L'oggetto di risposta ha due metodi per inviarla: *SendResponse* e *SendRedirect*. Si chiama *SendResponse* per mandare la risposta utilizzando il contenuto specificato e tutte le proprietà dell'intestazione dell'oggetto *TWebResponse*. Se basta reindirizzare il client Web a un altro URI, il metodo *SendRedirect* è più efficiente.

Se nessuno dei gestori di evento invia la risposta, l'oggetto Web application la invia quando il dispatcher ha finito. Tuttavia, se nessun elemento di azione segnala di aver

gestito la risposta, l'applicazione chiude la connessione con il client Web senza inviare nessuna risposta.

Generazione del contenuto dei messaggi di risposta

C++Builder fornisce una serie di oggetti per assistere gli elementi di azione nella produzione dei contenuti dei messaggi HTTP di risposta. Si possono usare per generare stringhe di comandi HTML che vengono salvati in un file o inviati direttamente al client Web. Si possono scrivere dei generatori di contenuti facendoli derivare da *TCustomContentProducer* o da uno dei suoi discendenti.

TCustomContentProducer fornisce un'interfaccia generica per la creazione di qualsiasi tipo MIME, come il contenuto di un messaggio di risposta HTTP. Tra i suoi discendenti vi sono produttori di pagine e produttori di tabelle:

- I produttori di pagine analizzano i documenti HTML cercando tag particolari che vengono sostituiti da un codice HTML ad hoc. Vengono illustrati nella prossima sezione.
- I produttori di tabelle creano comandi HTML in base alle informazioni presenti in un dataset. Sono descritti nel paragrafo "Utilizzo nelle risposte di informazioni di database" a pagina 33-18.

Utilizzo di componenti produttori di pagine

I produttori di pagine (*TPageProducer* e discendenti) prendono un modello HTML e lo convertono sostituendo tag speciali non visibili da HTML con codice personalizzato HTML. Si può memorizzare un insieme dei modelli risposta standard che vengono compilati dai produttori di pagine quando bisogna generare la risposta a un messaggio di richiesta HTTP. Si possono concatenare più produttori di pagine e costruire iterativamente un documento HTML con successivi affinamenti dei tag HTML trasparenti.

Modelli HTML

Un modello HTML è una sequenza di comandi HTML e di tag HTML trasparenti. Un tag HTML trasparente ha la forma

```
<#TagName Param1=Value1 Param2=Value2 ...>
```

Le parentesi angolari (< e >) definiscono l'intero campo d'azione del tag. Il simbolo (#) segue immediatamente la parentesi angolare aperta (<) senza spazi vuoti tra l'una e l'altra. Il simbolo # segnala al produttore di pagine che si tratta di un tag HTML trasparente. Il nome del tag segue immediatamente il simbolo #. Il nome del tag può essere qualsiasi identificativo valido e individua il tipo di conversione che il tag rappresenta.

Dopo il nome del tag, il tag HTML trasparente può includere facoltativamente parametri che specificano i dettagli della conversione da effettuare. Ogni parametro

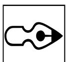
ha il formato *ParamName=Value*, senza spazi vuoti tra il nome del parametro, il simbolo (=) e il valore. I parametri sono separati da spazi vuoti.

Le parentesi angolari (< e >) rendono il tag trasparente ai browser HTML che non riconoscono il costrutto #TagName.

Benché sia possibile creare dei marcatori HTML-trasparenti personalizzati per rappresentare qualsiasi tipo di informazione trattato dal proprio produttore di pagine, esistono svariati nomi di marcatori predefiniti associati a valori del tipo dati *TTag*. Questi nomi di marcatori predefiniti corrispondono ai comandi HTML che è abbastanza probabile trovare nei messaggi di risposta. Questi nomi del tag predefiniti corrispondono a comandi HTML che facilmente variano in relazione ai messaggi di risposta e che sono riportati nella seguente tabella:

Nome del tag	Valore di TTag	In che cosa va convertito il tag
Link	*tgLink*	Collegamento ipertestuale. Il risultato è una sequenza HTML che inizia con il tag <A> e finisce con il tag </A>.
Image	*tgImage*	Immagine grafica. Il risultato è il tag HTML <IMG>.
Table	*tgTable*	Tabella HTML. Il risultato è una sequenza HTML che inizia con il tag <TABLE> e finisce con il tag </TABLE>.
ImageMap	*tgImageMap*	Immagine grafica con zone calde associate. Il risultato è una sequenza HTML che inizia con il tag <MAP> e finisce con il tag </MAP>.
Object	*tgObject*	Un oggetto incorporato ActiveX. Il risultato è una sequenza HTML che inizia con il tag <OBJECT> e finisce con il tag </OBJECT>.
Embed	*tgEmbed*	DLL di add-in conforme a Netscape -. Il risultato è una sequenza HTML che inizia con il tag <EMBED> e finisce con il tag </EMBED>.

Qualsiasi altro nome di marcatore viene associato a *tgCustom*. Il produttore di pagine non fornisce nessuna elaborazione intrinseca dei nomi di marcatore predefiniti. Il produttore di pagine non elabora in alcun modo i nomi di tag predefiniti, che servono solo per aiutare le applicazioni a organizzare il processo di conversione in molte delle attività più comuni.



I nomi di tag predefiniti non riconoscono la differenza tra maiuscole e minuscole.

Specifica del modello HTML

I produttori di pagine offrono molte possibilità per specificare il modello HTML. Si può dare alla proprietà *HTMLFile* il nome di un file contenente il modello HTML. Si può impostare la proprietà *HTMLDoc* a un oggetto *TStrings* contenente il modello HTML. Utilizzando per specificare il modello o la proprietà *HTMLFile* o la proprietà *HTMLDoc*, è possibile generare i comandi convertiti HTML chiamando il metodo *Content*.

Inoltre, è possibile chiamare il metodo *ContentFromString* per convertire direttamente un modello HTML passatogli come parametro sotto forma di una singola AnsiString. È possibile anche chiamare il metodo *ContentFromStream* per leggere il modello

HTML da un flusso. Così, ad esempio, si potrebbero memorizzare tutti i modelli HTML in un campo memo di un database e utilizzare il metodo *ContentFromStream* per ottenere i comandi convertiti HTML, leggendo il modello direttamente da un oggetto *TBlobStream*.

Conversione dei tag HTML trasparenti

Se si chiama uno dei suoi metodi *Content*, il produttore di pagine converte il modello HTML. Quando *Content* incontra un tag HTML trasparente, attiva l'evento *OnHTMLTag*. Per determinare il tipo di tag incontrato e sostituirlo con contenuto personalizzato è necessario scrivere un gestore di evento.

Non creando il gestore di evento *OnHTMLTag* del produttore di pagine, i tag HTML trasparenti sono sostituiti da una stringa vuota.

Utilizzo dei produttori di pagine da un elemento di azione

In un impiego tipico, un componente produttore di pagine utilizza la proprietà *HTMLFile* per specificare un file contenente un modello HTML. Il gestore di evento *OnAction* chiama il metodo *Content* per convertire il modello in una sequenza HTML definitiva:

```
void __fastcall WebModule1::MyActionEventHandler(TObject *Sender,
   TWebRequest *Request, TWebResponse *Response, bool &Handled)
{
  PageProducer1->HTMLFile = "Greeting.html";
  Response->Content = PageProducer1->Content();
}
```

Greeting.html è un file che contiene questo modello HTML:

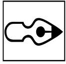
```
<HTML>
<HEAD><TITLE>Our Brand New Web Site</TITLE></HEAD>
<BODY>
Hello <#UserName>!  Welcome to our Web site.
</BODY>
</HTML>
```

Il gestore di evento *OnHTMLTag* sostituisce nell'HTML il tag ad hoc (<#UserName>) durante l'esecuzione:

```
void __fastcall WebModule1::HTMLTagHandler(TObject *Sender, TTag Tag,
   const AnsiString TagString, TStrings *TagParams, AnsiString &ReplaceText)
{
  if (CompareText(TagString,"UserName") == 0)
    ReplaceText = ((TPageProducer *)Sender)->Dispatcher->Request->Content;
}
```

Se il contenuto del messaggio di richiesta fosse la stringa *Mr. Ed*, il valore di *Response->Content* sarebbe

```
<HTML>
<HEAD><TITLE>Our Brand New Web Site</TITLE></HEAD>
<BODY>
Hello Mr. Ed!  Welcome to our Web site.
</BODY>
</HTML>
```



Questo esempio utilizza un gestore di evento *OnAction* per chiamare il produttore di contenuti e assegnare il contenuto del messaggio di risposta. Non è necessario scrivere un gestore di evento *OnAction* se la proprietà *HTMLFile* del produttore di pagine viene assegnata in fase di progettazione. In questo caso, basta assegnare *PageProducer1* come valore della proprietà *Producer* dell'elemento di azione per ottenere l'effetto ottenuto dal gestore di evento *OnAction* qui sopra.

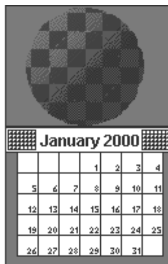
Concatenamento di produttori di pagine

Il testo di sostituzione di un gestore di evento *OnHTMLTag* non deve necessariamente essere la sequenza HTML da utilizzare nel messaggio HTTP di risposta. In certi casi, si utilizzano più produttori di pagine, nei quali l'output di uno è l'input del successivo.

Il modo più semplice per concatenare dei produttori di pagine è quello di associare ogni produttore di pagine a un elemento di azione separato, dove tutti gli elementi di azione hanno gli stessi *PathInfo* e *MethodType*. Il primo elemento di azione imposta il contenuto del messaggio di risposta Web dal suo produttore di contenuti, ma il suo gestore di evento *OnAction* si assicura che il messaggio non sia considerato gestito. L'elemento di azione successivo utilizza il metodo *ContentFromString* del suo produttore associato per trattare il contenuto del messaggio di risposta Web, e così via. Gli elementi di azione dopo il primo utilizzano un gestore di evento *OnAction* come il seguente:

```
void __fastcall WebModule1::Action2Action(TObject *Sender,
   TWebRequest *Request, TWebResponse *Response, bool &Handled)
{
  Response->Content = PageProducer2->ContentFromString(Response->Content);
}
```

Ad esempio, si consideri un'applicazione che restituisce pagine di calendario in risposta a messaggi di richiesta che specificano il mese e l'anno della pagina chiesta. Ogni pagina del calendario contiene un'immagine, seguita dal nome e dall'anno del mese fra piccole immagini del mese precedente e dei mesi prossimi, seguiti dal calendario vero e proprio. L'immagine che ne risulta si presenta così:



La scheda generale del calendario memorizzata in un file modello. Si presenta così:

```
<HTML>
<Head></HEAD>
<BODY>
<#MonthlyImage> <#TitleLine><#MainBody>
</BODY>
</HTML>
```

Il gestore di evento *OnHTMLTag* del primo produttore di pagine cerca il mese e l'anno dal messaggio di richiesta. Utilizzando quelle informazioni e il file modello, fa quanto segue:

- Sostituisce <#MonthlyImage> con <#Image Month=January Year=2000>.
- Sostituisce <#TitleLine> con <#Calendar Month=December Year=1999 Size=Small> January 2000 <#Calendar Month=February Year=2000 Size=Small>.
- Sostituisce <#MainBody> con <#Calendar Month=January Year=2000 Size=Large>.

Il gestore di evento *OnHTMLTag* del successivo produttore di pagine usa i contenuti prodotti dal primo produttore di pagine e sostituisce il marcatore <#*Image* Month=January Year=2000> con l'opportuno marcatore HTML <IMG>. Un ulteriore produttore di pagine risolve i tag #Calendar con tabelle appropriate HTML.

Utilizzo nelle risposte di informazioni di database

La risposta a un messaggio di richiesta HTTP può comprendere anche informazioni tratte da un database. Alcuni produttori di contenuti specializzati alla pagina Internet della Component palette possono generare il codice HTML capace di rappresentare in una tabella HTML i record tratti da un database.

Come approccio alternativo, certi componenti speciali contenuti nella pagina InternetExpress della Component palette consentono di realizzare Web server che fanno parte di un'applicazione database multi-tier. Per i dettagli "Creazione di applicazioni Web con InternetExpress" a pagina 29-33.

Aggiunta di una sessione a un modulo Web

Sia le applicazioni CGI di console che le applicazioni Win-CGI vengono lanciate in risposta a messaggi di richiesta HTTP. Quando in questi tipi di applicazioni si ha a che fare con un database, è possibile utilizzare la sessione predefinita per gestire le connessioni database, perché ogni messaggio di richiesta ha una propria istanza dell'applicazione. Ogni istanza dell'applicazione ha una propria sessione predefinita.

Scrivendo un'applicazione ISAPI o un'applicazione NSAPI, tuttavia, ogni messaggio di richiesta è gestito da un thread distinto di una singola istanza di applicazione. Per impedire alle connessioni database di vari thread di interferire l'una con l'altra, bisogna assegnare a ogni thread una propria sessione.

Ogni messaggio di richiesta in un'applicazione ISAPI o NSAPI produce un nuovo thread. Il modulo Web del thread in questione è generato dinamicamente in fase di esecuzione. Si aggiunge un oggetto *TSession* al modulo Web per gestire le connessioni database del thread contenente il modulo Web.

In fase di esecuzione, per ciascun thread viene generata un'istanza distinta del modulo Web. Ciascun modulo contiene l'oggetto sessione. Ciascuna sessione deve avere il proprio nome, in modo che i thread che gestiscono i diversi messaggi di richiesta non interferiscano con le connessioni dei database. Per fare in modo che gli

oggetti sessione in ogni modulo generino dinamicamente nomi univoci, si imposta la proprietà *AutoSessionName* dell'oggetto sessione. Ogni oggetto sessione genererà dinamicamente un proprio nome univoco e imposterà la proprietà *SessionName* di tutti i dataset nel modulo, in modo che facciano riferimento a quel nome univoco. Questo consente a ogni singolo thread di richiesta di interagire con il database senza interferire con gli altri messaggi di richiesta. Per ulteriori informazioni sule sessioni, consultare "Gestione delle sessioni di database" a pagina 24-17.

Rappresentazione di informazioni di database in HTML

I componenti produttori di contenuti specializzati della pagina Internet della Component palette dispongono di comandi HTML basati sui record di un dataset. Ci sono due tipi di produttori di contenuti associati ai dati:

- Il produttore di pagine di dataset, che formatta i campi di un dataset nel testo di un documento HTML.
- I produttori di tabelle, che formattano i record di un dataset come una tabella HTML.

Utilizzo dei produttori di pagine di dataset

I produttori di pagine di dataset funzionano come gli altri componenti produttori di pagine: convertono un modello comprendente tag HTML trasparenti in una rappresentazione finale HTML. Tuttavia, sono anche in grado di convertire i tag che hanno un tagname corrispondente al nome di un campo di un dataset nel valore corrente del campo. Per ulteriori informazioni sull'utilizzo di produttori di pagine in generale, consultare "Utilizzo di componenti produttori di pagine" a pagina 33-14.

Per utilizzare un produttore di pagine di dataset, aggiungere un componente *TDataSetPageProducer* al modulo Web e impostarne la proprietà *DataSet* al dataset i cui campi dovrebbero essere visualizzati nei contenuti HTML. Si crea un modello HTML che descrive l'output del produttore di pagine di dataset. Per ciascun valore di campo che si desidera visualizzare, si mette nel modello HTML un tag in formato

```
<#FieldName>
```

dove *FieldName* specifica il nome del campo nel dataset di cui si vuole visualizzare il valore.

Quando l'applicazione chiama il metodo *Content*, *ContentFromString* o *ContentFromStream*, il produttore di pagine di dataset sostituisce i valori del campo correnti con i tag che rappresentano campi.

Utilizzo di produttori di tabelle

La pagina Internet della Component palette contiene due componenti che creano tabelle HTML per rappresentare i record di un dataset:

- Produttori di tabelle di dataset, che formattano i campi di un dataset nel testo di un documento HTML.

- Produttori di tabelle di query, che eseguono una query dopo avere impostato i parametri forniti dal messaggio di richiesta e che formattano come tabella HTML il dataset risultante.

Utilizzando l'uno o l'altro dei due produttori di tabelle, è possibile adattare l'aspetto della tabella HTML risultante specificando le proprietà per il colore, il margine, lo spessore dei separatori e così via. Per impostare in fase di progetto le proprietà di un produttore di tabelle, fare doppio clic sul componente produttore di tabelle in modo da visualizzare la finestra di dialogo Response Editor.

Specifica degli attributi di tabella

Per rappresentare visivamente la tabella HTML che visualizza i record dal dataset i produttori di tabelle utilizzano l'oggetto *THTMLTableAttributes*. L'oggetto *THTMLTableAttributes* dispone di proprietà per fissare la larghezza e la spaziatura della tabella nel documento HTML per il colore di sfondo, lo spessore del bordo, il suo riempitivo e la sua spaziatura di cella. Queste proprietà sono trasformate in opzioni nel tag HTML <TABLE> creato dal produttore di tabelle.

In fase di progettazione, queste proprietà si specificano utilizzando l'*Object* Inspector. Selezionare l'oggetto produttore di tabella nell'*Object* Inspector ed espandere la proprietà *TableAttributes* per accedere alle proprietà di visualizzazione dell'oggetto *THTMLTableAttributes*.

È possibile anche specificare queste proprietà da programma in fase di esecuzione.

Specifica degli attributi di riga

Come nel caso degli attributi di tabella, si possono specificare l'allineamento e il colore di sfondo delle celle nelle righe della tabella che visualizzano dati. La proprietà *RowAttributes* è un oggetto *THTMLTableRowAttributes*.

In fase di progettazione, queste proprietà si specificano utilizzando l'*Object* Inspector ed estendendo la sua proprietà *RowAttributes*.

Impostando la proprietà *MaxRows*, si può anche regolare il numero di righe visualizzate di una tabella HTML.

Specifica delle colonne

Se in fase di progettazione si conosce il dataset della tabella, è possibile utilizzare il *Columns* editor per personalizzare i collegamenti e gli attributi di visualizzazione di campo delle colonne. Selezionare il componente produttore di tabelle e fare clic con il pulsante destro. Nel menu di scelta rapida, scegliere il *Columns* editor. Si possono così aggiungere, cancellare o riordinare le colonne nella tabella. È possibile impostare i collegamenti di campo e visualizzare le proprietà delle singole colonne nell'*Object* Inspector, dopo averle selezionate nel *Columns* editor.

Se il nome del dataset viene ottenuto tramite il messaggio di richiesta HTTP, è impossibile in fase di progettazione collegare i campi nel *Columns* editor. In fase di esecuzione, si possono però personalizzare le colonne mediante un programma configurando gli oggetti appropriati *THTMLTableColumn* e utilizzando i metodi della proprietà *Columns* per inserirli nella tabella. Se non si configura la proprietà *Columns*,

il produttore di tabella crea un set predefinito di colonne corrispondenti ai campi del dataset senza però impostare particolari caratteristiche di visualizzazione.

Inserimento di tabelle nei documenti HTML

Utilizzando le proprietà *Header* e *Footer* di un produttore di tabelle, si può incorporare la tabella HTML che rappresenta il dataset in un documento più grande. *Header* serve per specificare tutto ciò che viene prima della tabella e *Footer* per specificare tutto ciò che viene dopo la tabella.

Per creare i valori delle proprietà *Header* e *Footer* conviene utilizzare un altro produttore di contenuti (come un produttore di pagine).

Se la tabella viene inclusa in un documento più grande, conviene aggiungere un'intestazione. Allo scopo, si utilizzano le proprietà *Caption* e *CaptionAlignment*.

Impostazione di un produttore di tabelle di dataset

TDataSetTableProducer è un produttore di tabelle che crea una tabella HTML per un dataset. Per specificare il dataset contenente i record da visualizzare, si imposta la proprietà *DataSet* di *TDataSetTableProducer*. Non si imposta la proprietà *DataSource*, come si farebbe in un'applicazione database convenzionale con la maggior parte degli oggetti data-aware. Questo avviene perché *TDataSetTableProducer* genera il proprio datasource internamente.

È possibile impostare il valore di *DataSet* in progettazione se l'applicazione Web visualizza sempre record dallo stesso dataset. È necessario impostare la proprietà *DataSet* in esecuzione se si basa il dataset sulle informazioni nel messaggio di richiesta HTTP.

Impostazione di un produttore di tabelle di query

È possibile produrre una tabella HTML per visualizzare i risultati di una query, dove i parametri della query provengono dal messaggio di richiesta HTTP. Specificare l'oggetto *TQuery* che utilizza quei parametri come la proprietà *Query* di un componente *TQueryTableProducer*.

Se il messaggio di richiesta è una richiesta GET, i parametri della query vengono dai campi *Query* dell'URL che è stata data come destinazione del messaggio di richiesta HTTP. Se il messaggio di richiesta è una richiesta POST, i parametri della query vengono dal contenuto del messaggio di richiesta.

Quando si chiama il metodo *Content* di *TQueryTableProducer*, la query viene eseguita, utilizzando i parametri che trova nell'oggetto di richiesta. Quindi viene formattata una tabella HTML per visualizzare i record nel dataset risultante.

Come con qualsiasi produttore di tabella, è possibile personalizzare le proprietà dello schermo o i collegamenti di colonna della tabella HTML o incorporare la tabella in un documento HTML più grande.

Capitolo

34

Creazione di applicazioni Web Server con WebSnap

WebSnap potenzia Web Broker con componenti aggiuntivi, wizard e viste—rendendo più semplice la costruzione di applicazioni per Web server che distribuiscono pagine web data-driven complesse. Il supporto di WebSnap per più moduli e per lo scripting lato server server semplifica lo sviluppo e la manutenzione sia ai team di sviluppo in C++Builder sia ai Web designer.

WebSnap consente agli esperti di progettazione HTML all'interno del team di apportare un maggior contributo alla sviluppo e alla manutenzione dei server Web. Il prodotto finale del processo di sviluppo WebSnap include una serie di modelli di pagine HTML automatizzabili con script. Queste pagine possono essere modofcate utilizzando un editor HTML che supporti marcatori di script incorporati, come Microsoft FrontPage o anche un semplice editor di testi. Se occorre, le modifiche possono essere fatte ai modelli, anche dopo la distribuzione dell'applicazione. Non c'è alcuna necessità di modificare il codice sorgente del progetto, il che fa risparmiare prezioso tempo di sviluppo. Inoltre, il supporto per più moduli di WebSnap può essere utilizzato per suddividere l'applicazione in parti più piccole durante la fase di codifica del progetto. Gli sviluppatori in C++Builder possono lavorare in modo più indipendente.

I componenti dispatcher gestiscono automaticamente richieste di contenuti di pagina, sottomissioni di schede HTML, e richieste di immagini dinamiche. I componenti WebSnap, detti adapter, mettono a disposizione la capacità di definire un'interfaccia automatizzabile tramite script per le regole di gestione dell'applicazione. Ad esempio, l'oggetto *TDataSetAdapter* viene utilizzato per rendere i componenti dataset automatizzabili tramite script. È possibile utilizzare componenti produttore di WebSnap per costruire velocemente moduli e tabelle data-driven complesse oppure per utilizzare XSL per generare una pagina. È possibile utilizzare il componente session per tenere traccia degli utenti finali. È possibile utilizzare il componente user list per fornire l'accesso a nomi utente, password e diritti di accesso.

Web application wizard permette di costruire velocemente un'applicazione che sarà personalizzata con i componenti necessari. Web page module wizard permette di creare un modulo che definisce una nuova pagina nell'applicazione. Oppure utilizzare Web data module wizard per creare un contenitore per i componenti che vengono condivisi da tutta l'applicazione web.

Le viste modulo di pagina visualizzano il risultato degli script lato server senza eseguire l'applicazione. È possibile visualizzare l'HTML generato in un browser incorporato utilizzando la pagina Preview o sotto forma di testo utilizzando la pagina HTML Result. La pagina HTML Script visualizza la pagina con lo script lato server, utilizzata per generare l'HTML per la pagina.

Le seguenti sezioni di questo capitolo spiegano come si utilizzano i componenti WebSnap per creare un'applicazione per Web server.

Componenti fondamentali di WebSnap

Prima di potere costruire applicazioni per Web server utilizzando WebSnap, è necessario capire prima i componenti fondamentali utilizzati nello sviluppo WebSnap. Essi rientrano in tre categorie:

- Moduli Web, che contengono i componenti che preparano l'applicazione e definiscono pagine
- Adattatori, che forniscono un'interfaccia fra pagine HTML e l'applicazione per Web server stessa
- Produttori di pagina, che contengono le routine che creano le pagine HTML da servire all'utente finale

Le seguenti sezioni esaminano ogni tipo di componente più in dettaglio.

Moduli Web

I moduli Web sono il componente di base delle applicazioni WebSnap. Ogni applicazione server WebSnap deve avere almeno un modulo Web. Se necessario, è possibile aggiungerne altri. Esistono quattro tipi di moduli Web:

- Moduli pagina Web application (oggetti *TWebAppPageModule*)
- Moduli dati Web application (oggetti *TWebAppDataModule*)
- Moduli pagina Web (oggetti *TWebPageModule*)
- Moduli dati Web (oggetti *TWebDataModule*)

I moduli pagina web e i moduli pagina Web application forniscono i contenuti per le pagine web. I moduli dati Web e i moduli dati Web application agiscono come contenitori per i componenti condivisi all'interno dell'applicazione; nelle applicazioni WebSnap svolgono la stessa funzione svolta dai comuni moduli dati nelle normali applicazioni in C++ Builder. Nell'applicazione server è possibile includere un qualsiasi numero di moduli pagina Web o moduli dati. Ci si potrebbe

chiedere di quanti moduli ha bisogno l'applicazione Web. Ogni applicazione WebSnap ha bisogno di un modulo (e solo uno) Web application di qualche tipo. Oltre a quello, è possibile aggiungere tutti i moduli pagina Web o dati che sono necessari.

Nel caso dei moduli pagina Web, una buona regola empirica è quella di avere un modulo per ogni stile di pagina. Se si prevede di implementare una pagina che può utilizzare il formato di una pagina esistente, è possibile che non sia necessario un nuovo modulo pagina Web. Le modifiche a un modulo pagina esistente possono essere sufficienti. Se la pagina è molto diversa dai moduli esistenti, probabilmente si vorrà creare un nuovo modulo. Ad esempio, si supponga che si stia provando a costruire un server per gestire online le vendite per catalogo. Le pagine che descrivono i prodotti disponibili potrebbero condividere tutte lo stesso modulo pagina Web, poiché le pagine possono contenere tutte gli stessi tipi di informazioni di base utilizzando la stessa disposizione. Un modulo d'ordine, tuttavia, probabilmente richiederà un modulo pagina Web differente, poiché il formato e la funzione di un modulo d'ordine sono diversi rispetto a quelli di una pagina per la descrizione di un prodotto.

Le regole sono diverse nel caso dei moduli dati Web. I componenti che possono essere condivisi da molti moduli Web dovrebbero essere messi in un unico modulo dati Web in modo da semplificare l'accesso condiviso. Sarà opportuno anche mettere i componenti che possono essere utilizzati da molte applicazioni web nel rispettivo modulo dati Web. In questo modo è possibile condividere facilmente quegli elementi tra le applicazioni. Naturalmente, se nessuna di queste considerazioni è applicabile al proprio caso, si potrebbe anche decidere di non utilizzare affatto i moduli dati Web. Il loro utilizzo è analogo a quello dei normali moduli dati, e deve essere supportato del proprio giudizio e dalla propria esperienza.

Tipi di modulo Web application

I moduli Web application offrono un controllo centralizzato per le regole di gestione e per i componenti non visuali nell'applicazione web. I due tipi di moduli Web application sono riportati nella tabella seguente.

Tabella 34.1 Tipi di modulo Web application

Tipo di modulo Web application	Descrizione
Pagina	Crea una pagina di contenuti. Il modulo pagina contiene un produttore di pagine che è responsabile della generazione del contenuto di una pagina. Il produttore di pagine visualizza la pagina ad esso associata quando il pathinfo della richiesta HTTP corrisponde al nome della pagina. Se pathinfo è vuoto, la pagina può agire come pagina predefinita.
Dati	Utilizzato come un contenitore per componenti condivisi da altri moduli, come i componenti database utilizzati da più moduli pagina web.

I moduli Web application agiscono come contenitori per i componenti che eseguono funzioni per l'applicazione in generale — come la distribuzione delle richieste, la gestione delle sessioni e la manutenzione delle liste di utenti. Se si ha già una certa

familiarità con l'architettura Web Broker, è possibile pensare ai moduli Web application come se fossero simili a oggetti *TWebApplication*. I moduli Web application contengono anche le funzionalità di un normale modulo Web, sia esso pagina o dati, a seconda del tipo di modulo Web application. Il progetto può contenere solo un modulo Web application. In ogni caso non sarà mai necessario più di un modulo; è possibile aggiungere al server i normali moduli Web per fornire una qualsiasi funzione supplementare desiderata.

Utilizzare il modulo Web aplication per contenere le funzioni basilari dell'applicazione server. Se il server gestirà una home page di qualche sorta, si potrebbe utilizzare per il modulo Web application un *TWebAppPageModule* invece di un *TWebAppDataModule*, in modo da non dover creare un modulo pagina Web aggiuntivo per quella pagina.

Moduli pagina Web

Ad ogni modulo pagina Web è associato un produttore di pagine. Non appena riceve una richiesta, il dispatcher delle pagine analizza la richiesta e chiama il modulo pagina appropriato per elaborare la richiesta e restituire il contenuto della pagina.

Analogamente ai moduli dati Web, i moduli pagina Web sono contenitori per componenti. Tuttavia, un modulo pagina Web è molto di più di un mero contenitore. Un modulo pagina Web è utilizzato specificatamente per produrre una pagina web.

Tutti i moduli pagina Web hanno una vista editor, chiamata Preview, che permette di vedere in anteprima la pagina man mano che la si costruisce. È possibile sfruttare completamente l'ambiente visuale per lo sviluppo di applicazioni offerto da C++Builder.

Componente produttore di pagine

I moduli pagina Web hanno una proprietà che identifica il componente produttore di pagine responsabile della generazione dei contenuti della pagina. (Pe rmaggiori informazioni sui produttori di pagina, vedere "Produttori di pagina" on page 34-7.) WebSnap page module wizard aggiunge automaticamente un produttore quando si crea un modulo pagina Web. È possibile modificare in seguito il componente produttore di pagine prelevando un produttore diverso dalla tavolozza WebSnap. Se il modulo pagina ha un file di modello, assicurarsi tuttavia che il contenuto di questo file sia compatibile con il nuovo componente produttore.

Nome della pagina

I moduli pagina Web hanno un nome di pagina che può essere utilizzato per referenziare la pagina in una richiesta HTTP o all'interno della logica dell'applicazione. Una factory nella unit del modulo pagina Web specifica il nome della pagina per il modulo pagina Web.

Modello del produttore

La maggior parte dei produttori di pagine utilizzano un modello. Di solito i modelli HTML contengono del codice HTML statico frammisto a marcatori trasparenti o a script lato server. Quando i produttori di pagina creano il loro contenuto, essi sostituiscono i marcatori trasparenti con i valori appropriati ed eseguono lo script

lato server per produrre l'HTML che viene visualizzato da un browser client. (XSLPageProducer fa eccezione. Utilizza dei modelli XSL, che contengono XSL invece che HTML. I modelli XSL non supportano marcatori trasparenti o script lato server.)

Ai moduli pagina Web potrebbe essere associato un file di modello, gestito come parte della unit. Un file di modello gestito viene visualizzato nel Project Manager e ha gli stessi nome file e ubicazione di base del file di servizio della unit. Se il modulo pagina Web non ha un file di modello associato, le proprietà del componente produttore di pagine specificano il modello.

Moduli dati Web

Come i moduli dati standard, i moduli dati Web sono un contenitore per i componenti prelevati dalla tavolozza. I moduli dati forniscono una superficie di progettazione per aggiungere, rimuovere e selezionare i componenti. Il modulo dati Web è diverso rispetto a un modulo dati standard per la struttura della unit e delle interfacce che il modulo dati Web implementa.

Utilizzare il modulo dati Web come un contenitore per componenti che sono condivisi in tutta l'applicazione. Ad esempio, è possibile mettere un componente dataset in un modulo dati e accedere al dataset da:

- un modulo pagina che visualizza una griglia, e
- un modulo pagina che visualizza una scheda di immissione.

È possibile anche utilizzare i moduli dati Web per contenere set di componenti utilizzabili da varie applicazioni per Web server.

Struttura di una unit di un modulo dati Web

I moduli dati standard hanno una variabile detta variabile di scheda, utilizzata per accedere all'oggetto modulo dati. I moduli dati Web sostituiscono la variabile con una funzione, che è definita nella unit di un modulo dati Web e ha lo stesso nome del modulo dati Web. Lo scopo della funzione è identico a quello della variabile che sostituisce. Le applicazioni WebSnap possono essere multithread e possono avere più istanze di un particolare modulo per soddisfare più richieste contemporaneamente. Quindi, si utilizza la funzione per restituire l'istanza corretta.

La unit del modulo dati Web registra anche una factory per specificare in che modo il modulo dovrebbe essere gestito dall'applicazione WebSnap. Ad esempio, dei flag indicano se memorizzare nella cache il modulo e riutilizzarlo per altre richieste o distruggere il modulo una volta che è stata soddisfatta una richiesta.

Adattatori

Gli adattatori definiscono un'interfaccia di script per l'applicazione server. Essi permettono di inserire linguaggi di scripting all'interno della pagina e di reperire informazioni effettuando chiamate dal codice dello script agli adattatori. Ad esempio, è possibile utilizzare un adapter per definire i campi di dati da visualizzare in una pagina HTML. Una pagina HTML con script può quindi contenere dei contenuti HTML e istruzioni script che acquisiscono i valori di quei campi dati. Il

comportamento è simile a quello dei marcatori trasparenti utilizzati nelle applicazioni Web Broker. Gli adattatori supportano anche le azioni che eseguono comandi. Ad esempio, il fare clic su un collegamento ipertestuale o il confermare una pagina HTML può innescare azioni dell'adapter.

Gli adattatori semplificano la creazione di pagine HTML in modo dinamico. Utilizzando adattatori nell'applicazione, è possibile includere script orientati agli oggetti capaci di supportare una logica condizionale e i clicli. Senza adattatori e scripting lato server, è necessario scrivere nei gestori di evento C++molto più codice per la logica di generazione dell'HTML. L'utilizzo di adattatori può ridurre notevolmente i tempo di sviluppo.

Per maggiori dettagli sullo scripting, vedere pagina 34-33 e "Indirizzamento di richieste e risposte" a pagina 34-36.

Per creare i contenuti della pagina è possibile utilizzare quattro tipi di componenti adapter: campi, azioni, errori e record.

Campi

I campi sono componenti che il produttore di pagine utilizza per ottenere i dati dall'applicazione e visualizzare il contenuto in una pagina web. I campi possono essere utilizzati anche per acquisire un'immagine. In questo caso, il campo restituisce l'indirizzo dell'immagine scritta sulla pagina web. Quando una pagina visualizza il proprio contenuto, viene inviata una richiesta all'applicazione per Web server, la quale richiama il dispatcher dell'adapter per ottenere l'immagine effettiva dal componente campo.

Azioni

Le azioni sono componenti che eseguono comandi per conto dell'adapter. Quando un produttore di pagine genera la propria pagina, il linguaggio di scripting chiama i componenti azione dell'adapter per restituire il nome dell'azione insieme agli eventuali parametri necessari per eseguire il comando. Ad esempio, si immagini di fare clic su un pulsante in una pagina HTML per cancellare una riga da una tabella. L'operazione restituisce, nella richiesta HTTP, il nome dell'azione associata al pulsante e un parametro che indica il numero di riga. Il dispatcher dell'adapter individua il componente azione chiamato e passa all'azione il numero di riga sotto forma di parametro.

Errori

Gli adattatori mantengono un elenco degli errori che si verificano durante l'esecuzione di un'azione. I produttori di pagina possono accedere a questo elenco di errori e visualizzarli nella pagina web che l'applicazione restituisce all'utente finale.

Record

Alcuni componenti adapter, come *TDataSetAdapter*, rappresentano più record. L'adapter fornisce un'interfaccia di scripting che consente l'iterazione fra i record. Alcuni adattatori supportano la paginazione e iterare solo attraverso i record alla pagina corrente.

Produttori di pagina

I produttori di pagina sono utilizzati per generare i contenuti per conto di un modulo pagina Web. I produttori di pagina forniscono le seguenti funzionalità:

- Generano un contenuto HTML.
- Possono referenziare un file esterno utilizzando la proprietà HTMLFile, oppure la stringa interna utilizzando la proprietà HTMLDoc.
- Quando i produttori sono utilizzati con un modulo pagina Web, il modello può essere un file associato a una unit.
- I produttori generano dinamicamente il codice HTML che può essere inserito nel modello utilizzando i marcatori trasparenti o lo scripting attivo. I marcatori trasparenti possono essere utilizzati nello stesso modo delle applicazioni WebBroker. Per maggiori informazioni sull'utilizzo dei marcatori trasparenti, vedere "Conversione dei tag HTML trasparenti" a pagina 33-16. Il supporto per l'Active scripting permette di incorporare n ela pagina HTML JScript o VBScript.

Il metodo standard WebSnap per l'utilizzo dei produttori di pagina è il seguente. Quando si crea un modulo pagina Web, è necessario scegliere nel Web page module wizard un tipo di produttore di pagine. Vi sono molte possibilità, ma la maggior parte degli sviluppatori WebSnap crea il prototipo delle proprie pagine utilizzando un adapter produttore di pagine, *TAdapterPageProducer*. L'adapter produttore di pagine permette di creare un prototipo di pagina web utilizzando un processo analogo al modello a componenti standard. All'adapter produttore di pagine si aggiunge un tipo di scheda, una scheda di adapter. Poiché sono necessari, è possibile aggiungere alla scheda adapter componenti adapter (come griglie adapter). Utilizzando adattatori produttori di pagine, è possibile creare pagine web usando una tecnica simile a quella standard utilizzata da C++Builder per la costruzione delle interfacce utente.

Vi sono alcuni casi in cui è più opportuno passare da un adapter produttore di pagine a un normale produttore di pagine. Ad esempio, uno dei compiti di un adapter produttore di pagine è quello di generare dinamicamente in fase di esecuzione script in un modello di pagina. Si potrebbe decidere che uno script statico potrebbe ottimizzare il server. Inoltre, gli utenti esperti con il linguaggio di scripting potrebbero volere apportare modifiche direttamente nello script. In questo caso, si deve utilizzare un normale produttore di pagina per evitare conflitti tra script dinamico e statico. Per apprendere come passare a un normale produttore di pagine, vedere "Progettazione HTML evoluta" on page 34-25

È possibile anche utilizzare i produttori di pagina nello stesso modo in cui li si utilizzerebbe nelle applicazioni Web Broker, associando il produttore a un elemento di azione Web dispatcher. I vantaggi derivanti dall'utilizzo del modulo pagina Web sono

- la capacità di vedere in anteprima il layout della pagina senza dover eseguire l'applicazione, e
- la capacità di associare al modulo un nome di pagina, in modo che il dispatcher delle pagine possa chiamare automaticamente il produttore di pagine.

Creazione di applicazioni per Web server con WebSnap

Se si guarda il codice sorgente per WebSnap, si scoprirà che WebSnap comprende centinaia di oggetti. Infatti, WebSnap è così ricco di oggetti e funzioni che si potrebbe spendere molto tempo a studiarne in dettaglio l'architettura prima di capirlo completamente. Per fortuna, non è assolutamente necessario capire l'intero sistema WebSnap prima di cominciare a sviluppare l'applicazione server.

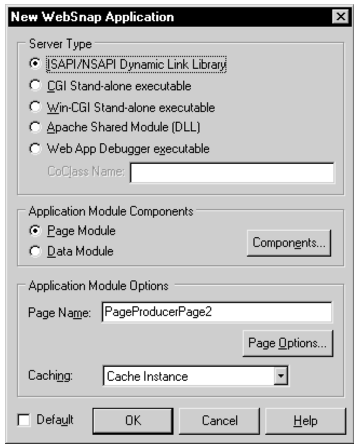
In questa sezione si apprenderanno molte cose sul funzionamento di WebSnap creando una nuova applicazione per Web server.

Per creare una nuova applicazione per Web server utilizzando l'architettura WebSnap:

1. Scegliere il comando File|New|Other.
2. Nella finestra di dialogo New Items, selezionare la pagina WebSnap e scegliere WebSnap Application.

   Viene visualizzata una finestra di dialogo (come mostrato nella Figure 34.1).
3. Specificare il tipo server corretto.
4. Utilizzare i componenti pulsante per specificare i componenti di modulo dell'applicazione.
5. Utilizzare il pulsante Page Options per selezionare le opzioni di modulo dell'applicazione.

Figura 34.1 Finestra di dialogo New WebSnap application



Selezione di un tipo server

Selezionare uno dei seguenti tipi di applicazione per Web server, a seconda del tipo di server Web dell'applicazione.

Tabella 34.2 Tipi di applicazione per Web server

Tipo server	Descrizione
ISAPI e NSAPI	Configura il progetto come una DLL con i metodi esportati attesi dal server Web.
Apache	Configura il progetto come una DLL con i metodi esportati attesi dal server Web Apache.
Indipendente CGI	Configura il progetto come un'applicazione per console, conforme allo standard Common Gateway Interface (CGI.
Indipendente Win-CGI	Configura il progetto come un'applicazione Windows CGI, una variante del CGI standard adattato per Windows.
Eseguibile App Debugger Web	Crea un ambiente per sviluppare e provare applicazioni per Web server. Questo tipo di applicazione non è destinato alla distribuzione.

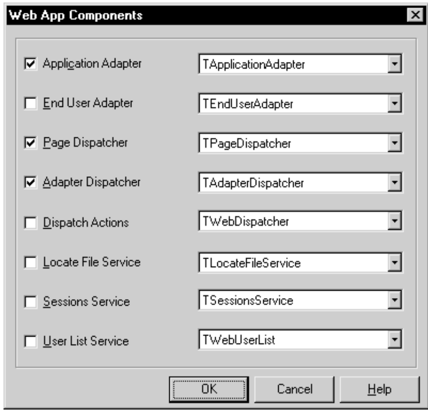
Specifica dei componenti del modulo applicazione

I componenti application forniscono le funzionalità dell'applicazione web. Ad esempio, l'inserimento di un componente adapter dispatcher gestisce automaticamente l'invio di schede HTML e la restituzione di immagini generate dinamicamente. L'inserimento di un dispatcher di pagine visualizza automaticamente il contenuto di una pagina quando il pathinfo della richiesta HTTP contiene il nome della pagina.

Se sulla finestra di dialogo New WebSnap application (vedere la Figure 34.1) si seleziona il pulsante Components viene visualizzata un'altra finestra di dialogo che

permette di selezionare uno o più dei componenti modulo Web application. La finestra di dialogo, detta finestra di dialogo Web App Components, è visualizzata nella Figure 34.2.

Figura 34.2 Finestra di dialogo Web App Component



La seguente tabella contiene una breve spiegazione dei componenti disponibili:

Tabella 34.3 Componenti Web application

Tipo di componente	Descrizione
Application Adapter	Contiene informazioni sull'applicazione, come il titolo. Il tipo predefinito è *TApplicationAdapter*.
End User Adapter	Contiene informazioni sull'utente, come il loro nome, i diritti di accesso e se essi sono connessi o meno. Il tipo predefinito è *TEndUserAdapter*. Si può anche selezionare *TEndUserSessionAdapter*.
Page Dispatcher	Esamina il pathinfo della richiesta HTTP e chiama il modulo pagina appropriato per restituire il contenuto di una pagina. Il tipo predefinito è *TPageDispatcher*.
Adapter Dispatcher	Gestisce automaticamente l'inoltro di schede HTML e le richieste di immagini dinamiche chiamando un adapter action e i componenti campo. Il tipo predefinito è *TAdapterDispatcher*.
Dispatch Actions	Permette di definire una raccolta di elementi di azione per gestire richieste in base al pathinfo e al tipo di metodo. Gli elementi di azione chiamano eventi definiti dall'utente o richiedono il contenuto di componenti page producer. Il tipo predefinito è *TWebDispatcher*.
Locate File Service	Fornisce il controllo sul caricamento dei file di modello e dei file di inclusione degli script mentra la Web application è in esecuzione. Il tipo predefinito è *TLocateFileService*.

Tabella 34.3 Componenti Web application

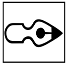
Tipo di componente	Descrizione
Sessions Service	Registra le informazioni sugli utenti necessarie per un breve periodo di tempo. Ad esempio, è possibile utilizzare le sessioni per tenere traccia di utenti collegati e disconnettere automaticamente un utente dopo un periodo di inattività. Il tipo predefinito è *TSessionsService*.
User List Service	Tiene traccia degli utenti autorizzati, delle loro password e dei loro diritti di accesso. Il tipo predefinito è *TWebUserList*.

Per ognuno dei precedenti componenti, i tipi di componenti elencati sono i tipi predefiniti forniti con C++Builder. Gli utenti possono creare i propri tipi di componenti o utilizzare tipi di componenti forniti da terze parti.

Selezione delle opzioni del modulo Web application

Se il tipo di modulo di applicazione selezionato è un modulo di pagina, è possibile associare alla pagina un nome immettendo un nome nel campo Page Name nella finestra di dialogo New WebSnap Application. In esecuzione, l'istanza di questo modulo può essere conservata in cache o rimossa dalla memoria una volta che la richiesta è stata sodisfatta. Selezionare dal campo Caching una delle due opzioni. È possibile selezionare altre opzioni del modulo pagina scegliendo il pulsante Page Options. Viene visualizzata la finestra di dialogo Application Module Page Options che presenta le seguenti categorie:

- Producer: Il tipo di produttore per la pagina può essere impostato ad *AdapterPageProducer*, *DataSetPageProducer*, *InetXPageProducer*, *PageProducer* o *XSLPageProducer*. Se il produttore di pagine selezionato supporta lo scripting, utilizzare l'elenco a discesa Script Engine per selezionare il linguaggio utilizzato per scrivere la pagina.



AdapterPageProducer supporta solo JScript.

- HTML: Se il produttore selezionato utilizza un modello HTML questo gruppo sarà visibile.
- XSL: Se il produttore selezionato utilizza un modello XSL, come *TXSLPageProducer*, questo gruppo sarà visibile.
- New File: Selezionare New File se si vuole che un file di modello sia creato e gestito come parte della unit. Un file di modello gestito viene visualizzato nel Project Manager e ha lo stesso nome di file e ubicazione del file sorgente della unit. Deselezionare New File se si vogliono utilizzare le proprietà del componente producer (di solito la proprietà *HTMLDoc* o *HTMLFile*).
- Template: Se New File è selezionato, scegliere il contenuto predefinito per il file di modello dalla caslla a discesa Template. Il modello standard visualizza il titolo dell'applicazione, il titolo della pagina e i collegamenti ipertestuali alle pagine pubblicate. Il modello vuoto crea una pagina vuota.
- Page: Immettere un nome di pagina e un titolo per il modulo pagina. Il nome della pagina viene utilizzato per referenziare la pagina in una richiesta HTTP o

all'interno della logica dell'applicazione, mentre il titolo è il nome che l'utente finale vedrà quando la pagina viene visualizzata in un browser.

- Published: Selezionare Published per far sì che la pagina risponda automaticamente alle richieste HTTP nel caso in cui il nome della pagina coincida con il pathinfo nel messaggio di richiesta.
- Login Required: Selezionare Login Required se si vuole che l'utente sia obbligato a immettere una password prima di poter accedere alla pagina.

A questo punto si è appreso come iniziare a creare un'applicazione server WebSnap. L'esercitazione WebSnap nella sezione successiva descrive come sviluppare un'applicazione più completa.

Esercitazione WebSnap

Questa esercitazione fornisce le istruzioni passo passo per la costruzione di un'applicazione WebSnap. L'applicazione WebSnap mostra come utilizzare i componenti HTML WebSnap per costruire un'applicazione che consente di modificare il contenuto di una tabella. Seguire le istruzioni dall'inizio alla fine. Se fosse necessaria una piccola pausa, è possibile utilizzare in qualsiasi momento il comando File|Save All per salvare il progetto.

Creazione della nuova applicazione

Questa sezione descrive come creare una nuova applicazione WebSnap di nome Country Tutorial. L'applicazione mostra agli utenti sul web una tabella di informazioni su diversi paesi. Gli utenti possono aggiungere e cancellare paesi e modificare le informazioni su paesi esistenti.

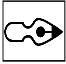
Fase 1. Avvio di WebSnap application wizard

1. Eseguire C++Builder e scegliere File|New|Other.
2. Nella finestra di dialogo New Items, selezionare la pagina WebSnap e scegliere WebSnap Application.
3. Nella finestra di dialogo New WebSnap Application:
   - Selezionare Web App Debugger executable.
   - Nel controllo di editing CoClass Name immettere CountryTutorial.
   - Selezionare Page Module come tipo di componente.
   - Nel campo Page Name scrivere Home.
4. Fare clic su OK.

Fase 2. Salvataggio dei file generati e del progetto

Per salvare il file unit Pascal e il progetto:

1 Selezionare File | Save All.
2 Nella finestra di dialogo Save, selezionare una directory appropriata in cui è possibile salvare tutti i file del progetto Country Tutorial.
3 Nel campo File name immettere HomeU.cpp e fare clic su Save.
4 Nel campo File name immettere CountryU.cpp e fare clic su Save.
5 Nel campo File name immettere CountryTutorial.bpr e fare clic su Save.



Salvare le unit e il progetto nella stessa directory perché l'applicazione cerca il file HomeU.html nella stessa directory dell'eseguibile.

Fase 3. Specifica del titolo dell'applicazione

Il titolo dell'applicazione è il nome visualizzato all'utente finale. Per specificare il titolo dell'applicazione:

1 Scegliere View | Project Manager.
2 Nel Project Manager espandere CountryTutorial.exe e fare doppio clic sulla voce HomeU.
3 Nella parte superiore della finestra di Object Inspector, selezionare dall'elenco a discesa ApplicationAdapter.
4 Nella pagina Properties, immettere Country Tutorial per la proprietà ApplicationTitle.
5 Fare clic sulla pagina Preview nella finestra dell'editor. Il titolo dell'applicazione viene visualizzato all'inizio della pagina con il nome della pagina, Home.

Questa pagina è molto scarna. È possibile migliorarla modificando il file HomeU.html, o utilizzando la pagina HomeU.html editor oppure utilizzando un editor esterno. Per ulteriori informazioni su come modificare il modello la pagina, vedere "Progettazione HTML evoluta" a pagina 34-25.

Creazione di una pagina CountryTable

Un modulo pagina Web definisce una pagina pubblicata e agisce anche come contenitore per i componenti dei dati. Ogni volta che una pagina web deve essere restituita all'utente finale, il modulo pagina Web estrae le informazioni necessarie dai componenti di dati in essa contenuti e utilizza quelle informazioni per creare una pagina.

In questa sezione si aggiungerà all'applicazione WebSnap un nuovo modulo pagina. Il modulo pagina aggiungerà all'applicazione Country Tutorial una seconda pagina visualizzabile. La prima pagina, Home, è stata definita quando si è creata l'applicazione. La seconda pagina, di nome countrytable, visualizza la tabella con le informazioni dei paesi.

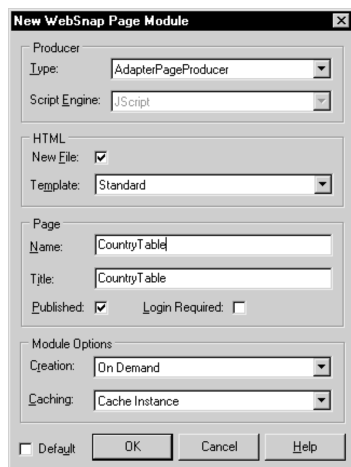
Fase 1. Aggiunta di un nuovo modulo pagina Web

Per aggiungere un nuovo modulo di pagina:

1 Scegliere File | New | Other.
2 Nella finestra di dialogo New Items, selezionare la pagina WebSnap, scegliere WebSnap Page Module e fare clic su OK.
3 Nella finestra di dialogo, impostare Producer Type ad AdapterPageProducer.
4 Nel campo Page Name immettere CountryTable. Notare che anche Title cambia man mano che si scrive.
5 Lasciare i campi rimanenti con i loro valori predefiniti.

   La finestra di dialogo dovrebbe assomigliare a quella mostrata nella Figure 34.3.
6 Fare clic su OK.

Figura 34.3 Finestra di dialogo New WebSnap Page Module per la pagina CountryTable



Il modulo CountryTable dovrebbe essere visualizzato nell'IDE. Dopo avere salvato il modulo, si aggiungeranno al modulo CountryTable i nuovi componenti.

Fase 2. Salvataggio del nuovo modulo pagina Web

Salvare la unit nella directory del file di progetto. Quando l'applicazione viene eseguita, cerca il file CountryTableU.html nella stessa directory dell'eseguibile.

1 Scegliere File | Save.
2 Immettere CountryTableU.cpp nel campo File name e fare clic su Save.

Aggiunta dei componenti dati al modulo CountryTable

TTable e *TDataSetAdapter* sono componenti data-aware; essi forniscono l'accesso a dati. *TTable* fornisce i dati per la tabella HTML. *TDataSetAdapter* permette allo script lato server di accedere al componente *TTable*. In questa sezione si aggiungeranno all'applicazione questi componenti data-aware.

Le fasi 1 e 2 seguenti presuppongono che si abbia una certa familiarità con la programmazione di database, anche se non è necessaria per completare questa esercitazione. WebSnap agisce solo come un'interfaccia (attraverso i componenti adapter) per i componenti database. Per maggiori informazioni sulla programmazione di database, è possibile fare riferimento alla Parte II di questo manuale.

Fase 1. Aggiunta di componenti data-aware

1. Scegliere View | Project Manager.
2. Nel Project Manager espandere CountryTutorial.exe e fare doppio clic sulla voce CountryTableU.
3. Scegliere View | Object TreeView. La finestra Object TreeView diventa attiva.
4. Selezionare la pagina BDE della Componente palette.
5. Selezionare un componente Table e aggiungerlo al modulo pagina Web CountryTable.
6. Selezionare un componente Session e aggiungerlo al modulo pagina Web CountryTable. Il componente Session è necessario perché si sta utilizzando un componente BDE (TTable) in un'applicazione multithread.
7. Selezionare il componente Session, chiamato Session1 per impostazione predefinita, nel modulo pagina Web o nell'Object TreeView. In questo modo si visualizzano nell'Object Inspector i valori del componente Session.
8. Nell'Object Inspector, impostare la proprietà *AutoSessionName* a **true**.
9. Selezionare il componente Table nel modulo pagina Web o nell'Object TreeView. In questo modo si visualizzano nell'Object Inspector i valori del componente Table.
10. Nell'Object Inspector, modificare le seguenti proprietà:
    - Impostare la proprietà *DatabaseName* a DBDEMOS.
    - Nella proprietà *Name* , immettere Country.
    - Impostare la proprietà *TableName* a country.db.
    - Impostare la proprietà *Active* a **true**.

Fase 2. Specifica di un campo chiave

Il campo chiave viene utilizzato per identificare i record all'interno di una tabella. Questo diventerà importante quando si aggiungerà all'applicazione una pagina di editing. Per specificare un campo chiave:

1. Nell'Object TreeView, espandere il nodo Session e DBDemos, quindi selezionare il nodo country.db. Questo nodo è il componente Country Table.
2. Fare clic destro sul nodo country.db e selezionare Fields Editor.
3. Fare clic destro sulla finestra CountryTable->Country editor e scegliere Add All Fields.

4 Selezionare il campo Name dall'elenco di campi aggiunti.

5 Nell'Object Inspector, espandere la proprietà *ProviderFlags*.

6 Impostare la proprietà *pfInKey* a **true**.

Fase 3. Aggiunta di un componente adapter

L'aggiunta dei componenti database è terminata. Ora, per esporre i dati nel TTable mediante script lato server, è necessario includere un componente dataset adapter (*TDataSetAdapter*). Per aggiungere un dataset adapter:

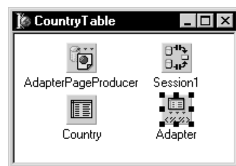
1 Scegliere il componente DataSetAdapter dalla pagina WebSnap della Component palette. Aggiungerlo al modulo Web CountryTable.

2 Modificare le seguenti proprietà nell'Object Inspector:

   - Impostare la proprietà *DataSet* a Country.
   - Impostare la proprietà *Name* ad Adapter.

   Una volta terminato, il modulo pagina Web CountryTable dovrebbe assomigliare a quello visualizzato nella Figure 34.4.

Figura 34.4 Modulo pagina Web CountryTable



Poiché gli elementi nel modulo non sono visuali, non è importate la posizione in cui appaiono nel modulo. Quello che importa è che il modulo contenga gli stessi componenti come mostrato in figura.

Creazione di una griglia per visualizzare i dati

Fase 1. Aggiunta di una griglia

Ora, aggiungere una griglia per visualizzare i dati dal database CountryTable:

1 Scegliere View | Project Manager.

2 Nel Project Manager, espandere CountryTutorial.exe e fare doppio clic sulla voce CountryTableU.

3 Scegliere View | Object TreeView o fare clic sull'Object TreeView.

4 Espandere il componente AdapterPageProducer. Questo componente genera lo script lato server per creare velocemente una tabella HTML.

5 Fare clic destro sulla voce WebPageItems e scegliere New Component.

6 Nella finestra di dialogo Add Web Component, selezionare AdapterForm, quindi fare clic su OK. Nell'Object TreeView viene visualizzato un componente AdapterForm1.

7 Fare clic destro su AdapterForm1 e selezionare New Component.

8 Nella finestra Add Web Component, selezionare AdapterGrid quindi fare clic su OK. Nell'Object TreeView viene visualizzato un componente AdapterGrid1.

9 Nella finestra Object Inspector, impostare la proprietà Adapter ad Adapter.

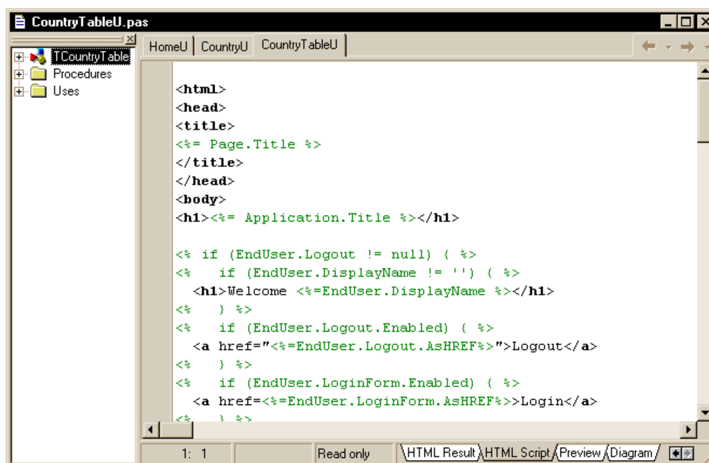
Per vedere un'anteprima della pagina, selezionare la pagina CountryTableU all'inizio dell'editor di codice, quindi selezionare la pagina Preview nella parte inferiore. Se la pagina Preview non viene visualizzata, utilizzare nella zona inferiore la freccia verso destra per far scorrere le pagine. L'anteprima dovrebbe assomigliare alla Figura 34.5.

Figura 34.5 Pagina CountryTable Preview



La pagina Preview mostra come verrebbe visualizzata la pagina finale HTML statica in un browser Web. Quella pagina è generata a partire da una pagina dinamica HTML che include script. Per vedere una rappresentanza testuale che mostra i comandi dello script, selezionare il separatore HTML Script nella zona inferiore della finestra dell'editor (vedere la Figura 34.6).

Figura 34.6 Pagina CountryTable HTML Script



La pagina HTML Script mostra una combinazione di HTML e script. HTML e script vengono differenziati nell'editor in base al colore e agli attributi del font. Per impostazione predefinita, i marcatori HTML vengono visualizzati in nero e grassetto, mentre i nomi dell'attributo HTML vengono visualizzati in nero e i valori dell'attributo HTML vengono visualizzati in blu. Il codice dello script, che è visualizzato tra le parentesi angolari di script `<% %>`, è colorato in verde. È possibile modificare il colore predefinito del font e gli attributi di questi elementi nella pagina Color della finestra di dialogo Properties editor, che può essere visualizzata facendo clic destro sull'editor e selezionare Properties.

Vi sono altre due pagine dell'editor relative all'HTML. La pagina HTML Result visualizza il contenuto HTML non formattato dell'anteprima. Notare che le viste HTML Result, HTML Script e Preview sono tutte a sola lettura. È possibile utilizzare l'ultima pagina dell'editor relativa all'HTML, CountryTable.html, per le modifiche.

Se si desidera migliorare l'aspetto di questa pagina, è possibile aggiungere del codice HTML utilizzando in qualsiasi momento o la pagina CountryTable.html o un editor esterno. Per ulteriori informazioni su come modificare il modello di pagina, vedere "Progettazione HTML evoluta" a pagina 34-25.

Fase 2. Aggiunta alla griglia di comandi per l'editing

Gli utenti possono avere la necessità di aggiornare il contenuto della tabella cancellando, inserendo o modificando una riga. Per consentire agli utenti di fare tali aggiornamenti, aggiungere dei componenti comando.

Per aggiungere dei componenti comando:

1. Nell'Object TreeView per il CountryTable, espandere il componente AdapterPageProducer e tutti i suoi rami.
2. Fare clic destro sul componente AdapterGrid1 e scegliere Add All Columns. Al gruppo di adattatori vengono aggiunte cinque colonne.

3 Fare nuovamente clic destro sul componente AdapterGrid1 e scegliere New Component.

4 Selezionare AdapterCommandColumn e quindi fare clic su OK. Al componente AdapterGrid1viene aggiunta una voce AdapterCommandColumn1.

5 Fare clic destro su AdapterCommandColumn1 e scegliere Add Commands.

6 Selezionare contemporaneamente i comandi DeleteRow, EditRow e NewRow, quindi fare clic su OK.

7 Per vedere l'anteprima della pagina, fare clic sulla pagina Preview in fondo all'editor di codice. Alla fine di ogni riga nella tabella appaiono ora tre nuovi pulsanti (DeleteRow, EditRow e NewRow), come mostrato nella Figura 34.7. Quando l'applicazione è in esecuzione, la pressione di uno di questi pulsanti esegue l'azione associata.

Figura 34.7 CountryTable Preview dopo avere aggiunto i comandi di editing



Fare clic sul pulsante Save All per salvare l'applicazione prima di continuare.

Aggiunta di una scheda per l'editing

A questo punto è possibile creare un modulo pagina Web per gestire la scheda Edit per la tabella dei paesi. Gli utenti saranno in grado di modificare i dati nell'applicazione CountryTutorial mediante la scheda Edit. Quando l'utente preme i pulsanti EditRow o NewRow, viene visualizzata appositamente una scheda Edit. Una volta che l'utente ha terminato le operazioni con la scheda Edit, le informazioni modificate saranno visualizzate automaticamente nella tabella.

Fase 1. Aggiunta di un nuovo modulo pagina Web

Per aggiungere un nuovo modulo pagina Web:

1 Scegliere File | New | Other.
2 Nella finestra di dialogo New Items, selezionare la pagina WebSnap e scegliere WebSnap Page Module.
3 Nella finestra di dialogo, impostare l'opzione Producer Type ad AdapterPageProducer.
4 Impostare il campo Page Name a CountryForm.
5 Deselezionare la casella Published, in modo che questa pagina non venga visualizzata in un elenco di pagine disponibili su questo sito. Alla scheda Edit si accede mediante il pulsante Edit e i suoi contenuti dipendono dalla riga della tabella dei paesi che deve essere modificato.
6 Lasciare immutati i valori predefiniti proposti per i campi rimanenti e per le altre opzioni. Fare clic su OK.

Fase 2. Salvataggio del nuovo modulo

Salvare il modulo nella stessa directory del file di progetto. Quando si esegue l'applicazione, essa cerca il file CountryFormU.html nella stessa directory dell'eseguibile.

1 Scegliere File | Save.
2 Immettere CountryFormU.cpp nel campo File name e fare clic su OK.

Fase 3. Come rendere CountryTableU accessibile al nuovo modulo

Aggiungere la unit CountryTableU alla direttiva include per consentire al componente Adapter l'accesso al modulo.

1 Scegliere File | Include Unit Hdr.
2 Scegliere CountryTableU dall'elenco quindi fare clic su OK.
3 Scegliere File | Save.

Fase 4. Aggiunta dei campi di immissione

Aggiungere dei componenti al componente AdapterPageProducer per generare nella scheda HTML dei campi di immissione dati.

Per aggiungere dei campi di immissione:

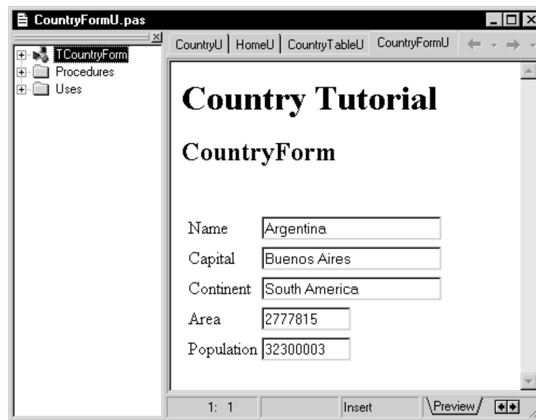
1 Scegliere View | Project Manager.
2 Nella finestra Project Manager, espandere CountryTutorial.exe e fare doppio clic sull'elemento CountryFormU.
3 Nell'Object TreeView espandere il componente AdapterPageProducer, fare clic destro su WebPageItems e selezionare New Component.
4 Selezionare AdapterForm, quindi fare clic su OK. Nell'Object TreeView viene visualizzato un elemento AdapterForm1.
5 Fare clic destro su AdapterForm1 e selezionare New Component.

6 Selezionare AdapterFieldGroup quindi fare clic su OK. Nell'Object TreeView viene visualizzato un elemento AdapterFieldGroup1.

7 Nella finestra Object Inspector, impostare la proprietà Adapter a CountryTable->Adapter. Impostare la proprietà *AdapterMode* a Edit.

8 Per vedere l'anteprima della pagina, fare clic sulla pagina Preview in fondo all'editor del codice. L'anteprima dovrebbe somigliare a quella visualizzata nella Figure 34.8.

Figura 34.8 Anteprima il CountryForm



Fase 5. Aggiunta di pulsanti

Aggiungere dei componenti al componente AdapterPageProducer per generare nella scheda HTML i pulsanti per l'inoltro. Per aggiungere i componenti:

1 Nell'Object TreeView espandere il componente *AdapterPageProducer* e tutti i suoi rami.

2 Fare clic destro su AdapterForm1 e scegliere New Component.

3 Selezionare AdapterCommandGroup quindi fare clic su OK. Nell'Object TreeView viene visualizzato un elemento AdapterCommandGroup1 a.

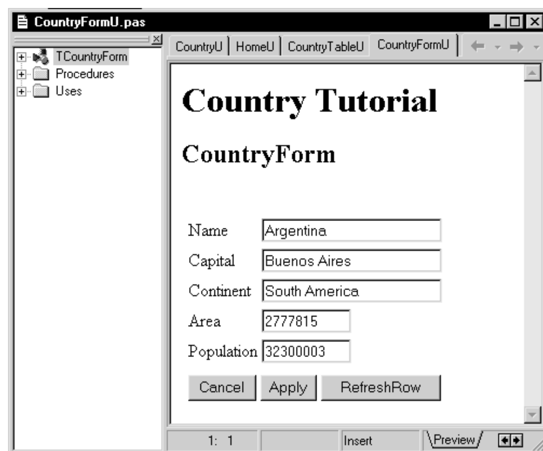
4 Nell'Object Inspector, impostare la proprietà *DisplayComponent* ad AdapterFieldGroup1.

5 Fare clic destro sull'elemento AdapterCommandGroup1 e scegliere Add Commands.

6 Selezionare contemporaneamente i comandi Cancel, Apply e Refresh Row, quindi fare clic su OK.

7 Per vedere l'anteprima della pagina, fare clic sulla pagina Preview in fondo alla finestra dell'editor del codice. Se l'anteprima non visualizza la pagina CountryForm, fare clic sulla pagina Code e quindi fare nuovamente clic sulla pagina Preview. L'anteprima dovrebbe somigliare a quella visualizzata nella Figure 34.9.

Figura 34.9 CountryForm con i pulsanti di inoltro



Fase 6. Collegamento di azioni della scheda con la pagina della griglia

Quando l'utente fa clic su un pulsante, viene eseguita un'azione dell'adapter il quale a sua volta esegue l'azione descritta. Per apecificare quale pagina visualizzare dopo l'esecuzione dell'azione dell'adapter:

1. Nell'Object TreeView, espandere AdapterCommandGroup1 per visualizzare le voci CmdCancel, CmdApply e CmdRefreshRow.
2. Selezionare CmdCancel. Nella finestra dell'Object Inspector, scrivere CountryTable nella proprietà *PageName*.
3. Selezionare CmdApply. Nella finestra dell'Object Inspector, scrivere CountryTable nella proprietà *PageName*.

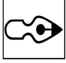
Fase 7. Collegamento delle azioni della griglia con la pagina della scheda

Premendo un pulsante nella griglia viene eseguita un'azione dell'adapter. Per specificare quale pagina deve essere visualizzata in risposta all'azione dell'adapter:

1. Scegliere View | Project Manager.
2. Nel Project Manager, espandere CountryTutorial.exe e fare doppio clic sulla voce CountryTableU.
3. Nell'Object TreeView, espandere il componente *AdapterPageProducer* e tutti i suoi rami per visualizzare le voci CmdNewRow, CmdEditRow e CmdDeleteRow. Queste voci vengono visualizzate sotto la voce AdapterCommandColumn1.
4. Selezionare CmdNewRow. Nell'Object Inspector, immettere CountryForm nella proprietà *PageName*.
5. Selezionare CmdEditRow. Nell'Object Inspector, immettere CountryForm nella proprietà *PageName*.

Per verificare che l'applicazione funzioni e che tutti i pulsanti eseguano qualche azione, eseguire l'applicazione. Per apprendere come eseguire l'applicazione, vedere

"Esecuzione dell'applicazione ultimata" a pagina 34-23. Quando si esegue l'applicazione, si manda in esecuzione un server. Per controllare che l'applicazione funzioni, è necessario visualizzarlo in un browser Web. È possibile fare ciò avviandolo dal Web Application debugger.



Non sarà visualizzata alcuna indicazione relativa a errori di database, come un tipo non valido. Ad esempio, provare ad aggiungere nel campo Area un nuovo paese con un valore non ammesso (ad esempio, 'abc.

Esecuzione dell'applicazione ultimata

Per eseguire l'applicazione ultimata:

1 Scegliere Run | Run. Viene visualizzata una scheda. Le Web Application compilate come eseguibili di Web App Debugger sono server COM e la scheda che si vede è la finestra console del server COM. La prima volta che si esegue il progetto, esso registra l'oggetto COM a cui Web App Debugger può accedere direttamente.
2 Scegliere Tools | Web App Debugger.
3 Fare clic sul link dello URL predefinito per visualizzare la pagina ServerInfo. La pagina ServerInfo visualizza i nomi di tutti gli eseguibili Web Application Debugger registrati.
4 Scegliere nell'elenco a discesa CountryTutorial e fare clic sul pulsante Go.

Il browser visualizzerà l'applicazione Country Tutorial. Fare clic sul link CountryTable per vedere la pagina CountryTable.

Aggiunta della notifica degli errori

A questo punto dello sviluppo, l'applicazione non visualizza all'utente alcun errore. Ad esempio, se si immettono delle lettere nel campo Area del record di un paese, non verrà visualizzato alcun avvertimento del fatto che si è verificato un errore. In questa sezione si aggiungerà un componente AdapterErrorList per visualizzare gli errori che si verificano durante l'esecuzione delle azioni dell'adapter che modificano la tabella dei paesi.

Fase 1. Aggiunta alla griglia del supporto per gli errori

Per aggiungere alla griglia il supporto per gli errori:

1 Nell'Object TreeView di CountryTable, espandere il componente *AdapterPageProducer* e tutti i suoi rami in modo da visualizzare AdapterForm1.
2 Fare clic destro su AdapterForm1 e scegliere New Component.
3 Selezionare dall'elenco AdapterErrorList, quindi fare clic su OK. Nell'Object TreeView viene visualizzato un elemento AdapterErrorList1.
4 Spostare AdapterErrorList1 prima di AdapterGrid1 (o trascinandolo o utilizzando la freccia verso l'alto nella barra strumenti dell'Object TreeView).

5 Nell'Object Inspector, impostare la proprietà *Adapter* ad Adapter.

Fase 2. Aggiunta alla scheda del supporto per gli errori

Per aggiungere alla scheda il supporto per gli errori:

1 Nell'Object TreeView di CountryForm, espandere il componente *AdapterPageProducer* e tutti i suoi rami in modo da visualizzare AdapterForm1.

2 Fare clic destro su AdapterForm1 e scegliere New Component.

3 Selezionare dall'elenco AdapterErrorList, quindi fare clic su OK. Nell'Object TreeView viene visualizzato un elemento AdapterErrorList1.

4 Spostare AdapterErrorList1 prima di AdapterFieldGroup1 (o trascinandolo o utilizzando la freccia verso l'alto nella barra strumenti dell'Object TreeView).

5 Nell'Object Inspector, impostare la proprietà *Adapter* a CountryTable->Adapter.

Fase 3. Verifica del meccanismo di segnalazione errori

Per osservare il meccanismo di segnalazione errori, è necessario fare prima una piccola modifica nell'IDE di C++Builder. Selezionare Tools | Debbugger Options. Nella pagina Language Exceptions, verificare che la casella Stop on Delphi Exceptions non sia selezionata, cosa che permetterà all'applicazione di continuare nel caso vengano rilevate eccezioni. A questo punto, per verificare gli errori della griglia:

1 Eseguire l'applicazione e, utilizzando Web Application Debugger, passare alla pagina CountryTable. Per informazioni sull'operazione, vedere "Esecuzione dell'applicazione ultimata" a pagina 34-23.

2 Aprire un'altra finestra del browser e passare alla pagina CountryTable.

3 Fare clic sul pulsante DeleteRow sulla prima riga della griglia.

4 Senza aggiornare il secondo browser, fare clic sul pulsante DeleteRow sulla prima riga della griglia.

Prima della griglia viene viusalizzato il messaggio di errore "Row not found in Country".

Per verificare gli errori di scheda:

1 Eseguire l'applicazione e, utilizzando Web Application debugger, passare alla pagina CountryTable.

2 Fare clic sul pulsante EditRow. Viene visualizzata la pagina CountryForm.

3 Modificare il campo Area in 'abc' e fare clic sul pulsante Apply.

Sopra al primo campo verrà visualizzato il messaggio di errore ("Invalid value for field 'Area'".

A questo punto l'esercitazione WebSnap è stata completata. Prima di continuare si potrebbe voler selezionare di nuovo la casella di controllo Stop on Delphi Exceptions. Inoltre, salvare l'applicazione scegliendo File | Save All in modo che l'applicazione ultimata sia disponibile e consultabile in futuro.

Progettazione HTML evoluta

Utilizzando gli adattatori e i produttori di pagina di adapter, WebSnap semplifica la creazione di pagine HTML con supporto di script nell'applicazione per Web server. Utilizzando gli strumenti di WebSnap è possibile creare un'interfaccia Web per i dati dell'applicazione, in grado di soddisfare tutte le proprie esigenze. Una potente funzione di WebSnap, tuttavia, è la capacità di incorporare nell'applicazione le capacità di progettazione per Web di altre sorgenti. Questa sezione tratta alcune strategie per espandere il processo di progettazione e di manutenzione del Web server includendo altri strumenti e membri del team non programmatori.

I prodotti finali per lo sviluppo WebSnap sono l'applicazione server e i modelli HTML per le pagine prodotte dal server. I modelli includono una miscela di scripting e di HTML. Una volta generati nella fase iniziale, possono essere modificati in qualunque momento utilizzando un qualsiasi strumento HTML. (Sarebbe meglio utilizzare uno strumento che supporti marcatori di script incorporati, come Microsoft FrontPage, per assicurarsi che l'editor non danneggi casualmente lo script). La capacità di modificare le pagine dei modelli esternamente all'IDE può essere utilizzata in vari modi.

Ad esempio, gli sviluppatori C++ Builder possono modifcare i modelli HTML in progettazione utilizzando un qualsiasi editor esterno. Ciò permette per utilizzare funzioni di formattazione e di scripting evolute che possono essere presenti in un editor HTML esterno ma non in C++ Builder. Per attivare un editor HTML esterno dall'IDE, fare quanto segue:

1 Dal menu principale, selezionare Tools | Environment Options. Nella finestra di dialogo Environment Options, fare clic sulla pagina Internet.
2 Nel riquadro Internet File Types, selezionare HTML e fare clic sul pulsante Edit per visualizzare la finestra di dialogo Edit Type.
3 Nel riquadro Edit Action, selezionare un'azione associata all'editor HTML. Ad esempio, per selezionare l'editor HTML predefinito nel sistema, scegliere Edit dall'elenco a discesa. Fare clic su OK due volte per chiudere le finestre di dialogo Edit Type e Environment Options.

Per modificare un modello HTML, aprire la unit che contiene quel modello. Nella finestra Edit, fare clic destro e selezionare Html Editor dal menu contestuale. L'editor HTML visualizza il modello da modificare in una finestra differente. L'editor viene eseguito in modo indipendente dell'IDE; una volta terminato, salvare il modello e chiudere l'editor.

Dopo la distribuzione del prodotto, si potrebbe voler modificare l'aspetto delle pagine HTML definitive. È probabile che il team di sviluppo software non sia nemmeno responsabile del formato della pagina finale. Tale compito potrebbe essere affidato, per esempio, a un Web page designer nell'organizzazione. È possibile che i Web page designer non abbiano alcuna esperienza di sviluppo in C++ Builder. Fortunatamente, non è necessario che ne abbiano. Essi possono modificare i modelli di pagina in qualsiasi punto del ciclo di sviluppo e di manutenzione del prodotto, senza mai modificare il codice sorgente. In tale modo, i modelli HTML di WebSnap possono rendere più efficienti lo sviluppo e la manutenzione del server.

Trattamento dello script lato server nei file HTML

Il codice HTML nei modelli di pagina può essere modificato in qualsiasi momento del ciclo di sviluppo. Tuttavia, lo scripting lato server deve essere affrontato in modo diverso. È sempre possibile modificare lo script lato server incluso nei modelli esternamente a C++ Builder, ma non è una cosa consigliabile per le pagine generate da un adapter produttore di pagine. L'adapter produttore di pagine è diverso dai normali produttori di pagine per il fatto che, in esecuzione, può modificare lo scripting lato server incluso nei modelli di pagina. Può essere difficile predire come agirà lo script se un altro script viene aggiunto dinamicamente. Se si vogliono trattare direttamente gli script, accertarsi che il modulo pagina Web contenga un produttore di pagine e non un adapter produttore di pagine.

Se si ha un modulo pagina Web che utilizza un adapter produttore di pagine, è possibile convertirlo in modo che utilizzi invece un normale produttore di pagine nel seguente modo:

1. Copiare nel modulo che si desidera convertire (supponendo che si chiami ModuleName) tutte le informazioni dalla pagina HTML Script sulla pagina ModuleName.html, sostituendo tutte le informazioni che precedentemente conteneva.
2. Trascinare un produttore di pagine (presente sulla pagina Internet della Component palette) sul modulo pagina Web.
3. Impostare la proprietà *ScriptEngine* del produttore di pagine in modo che corrisponda a quello dell'adapter produttore di pagine di cui prenderà il posto.
4. Modificare il produttore di pagine nel modulo pagina Web dall'adapter produttore di pagine nel nuovo produttore di pagine. Fare clic sulla pagina Preview per verificare che i contenuti della pagina restino invariati.
5. A questo punto l'adapter produttore di pagine è stato scavalcato. È possibile cancellarlo dal modulo pagina Web.

Supporto per il login

Molte applicazioni per Web server richiedono il supporto per il login. Ad esempio, un'applicazione server può richiedere che un utente esegua il login prima di concedere l'accesso ad alcune parti di un sito Web. Le pagine possono avere un aspetto diverso per i diversi utenti; il login può essere necessario per permettere al server Web di inviare le pagine corrette. Inoltre, poiché i server hanno limitazioni fisiche nei cicli di memoria e di processore, le applicazioni server hanno bisogno talvolta della capacità di limitare il numero di utenti in un dato momento.

Con WebSnap, l'inclusione nell'applicazione per Web server del supporto per il login è abbastanza semplice e diretta. In questa sezione, si apprenderà come è possibile aggiungere il supporto per il login, sia progettandolo dall'inizio del processo di sviluppo sia aggiungendolo a un'applicazione esistente.

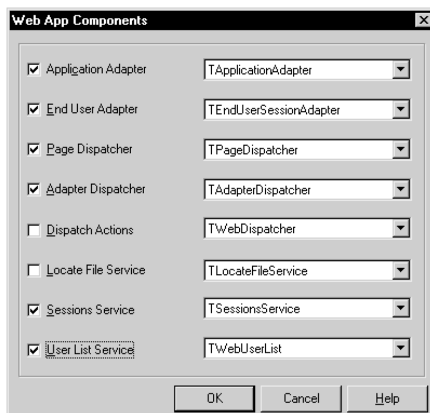
Aggiunta del supporto per il login

Per implementare il supporto per login, è necessario assicurarsi che il modulo Web application abbia i seguenti componenti:

- Un servizio di elenco utenti (un oggetto di tipo *TWebUserList*), che contiene i nomi degli utente, le password e i permessi accordati agli utenti del server
- Un servizio per le sessioni (*TSessionsService*), che memorizza le informazioni sugli utenti attualmente connessi al server
- Un adapter per utente finale (*TEndUserSessionAdapter*) che gestisce le azioni associate alla procedura di login

Quando si comincia a creare l'applicazione per Web server, è possibile aggiungere questi componenti utilizzando la finestra di dialogo New WebSnap Application. Fare clic sul pulsante Components su quella finestra di dialogo per visualizzare la finestra di dialogo New Web App Components. Selezionare le caselle End User Adapter, Sessions Service e Web User List. Nel menu a discesa accanto alla casella End User Adapter, selezionare *TEndUserSessionAdapter* in modo da selezionare il tipo di adapter per utente finale. (la scelta predefinita, *TEndUserAdapter*, non è appropriata per il supporto del login perché non può tracciare l'utente corrente). Una volta terminato, la finestra di dialogo dovrebbe assomigliare a quello visualizzata di seguito. Fare clic su OK due volte per chiudere le finestre di dialogo. Il modulo Web application web ora ha i componenti necessari per il supporto del login.

Figura 34.10 Finestra di dialogo Web App Components con le opzioni per supporto del login selezionate



Se si aggiunge il supporto per il login a un modulo Web application esistente, è possibile trascinare questi componenti direttamente nel modulo prelevandoli dalla pagina WebSnap della Component palette. Il modulo Web application si configurerà automaticamente.

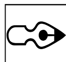
Il servizio per le sessioni e l'adapter per utente finale potrebbero non richiedere particolare attenzione durante la fase di progettazione a diffrenza invece dell'elenco utenti Web. È possibile aggiungere utenti predefiniti e impostarne i diritti di lettura/modifica tramite l'editor del componente WebUserList. Fare doppio clic sul

componente per visualizzare un editor che permette di impostare i nomi degli utenti, le password e i diritti di accesso. Per ulteriori informazioni su come configurare i diritti di accesso, vedere “Diritti di accesso degli utenti” on page 34-31.

Utilizzo del servizio per le sessioni

Il servizio per le sessioni, che è un oggetto del tipo *TSessionsService*, tiene traccia degli utenti registrati nell’applicazione per Web server. Il servizio per le sessioni si incarica di assegnare una sessione diversa ad ogni utente e di associare a un utente coppie di tipo nome/valore (come un nome utente).

Le informazioni contenute in un servizio per le sessioni sono conservate nella memoria dell’applicazione. Pertanto, l’applicazione per Web server deve essere tenuta in esecuzione fra le varie richieste affinché il servizio per le sessioni funzioni correttamente. Alcuni tipi di applicazioni server, come le CGI, terminano tra una richiesta e l’altra.



Se si vuole che l’applicazione supporti il login, assicurarsi di utilizzare un tipo di server che non termini tra un richiesta e l’altra. Se il progetto produce un eseguibile Web App debugger, è necessario fare in modo che l’applicazione venga eseguita in background prima che riceva una richiesta di pagina. In caso contrario terminerà dopo ogni richiesta di pagina e gli utenti non saranno mai in grado di ottenere la pagina di login.

Nel servizio per le sessioni ci sono due proprietà importanti che è possibile utilizzare per modificare il comportamento predefinito del server. La proprietà *MaxSessions* specifica quanti utenti possono essere registrati nel sistema in un dato momento. Il valore predefinito per *MaxSessions* è -1, che non pone alcuna limitazione software al numero di utenti consentiti. Naturalmente, l’hardware del server potrebbe disporre di poca memoria o servire gli utenti in modo ciclico, il che può influire negativamente sulle prestazioni del sistema. Se si pensa che il numero eccessivo di utenti potrebbe sovraccaricare il server, accertarsi di impostare *MaxSessions* a un valore appropriato.

La proprietà *DefaultTimeout* specifica il periodo di timeout predefinito in minuti. Dopo che sono trascorsi *DefaultTimeout* minuti senza alcuna attività da parte dell’utente, la sessione viene terminata automaticamente. Se l’utente aveva eseguito il login, tutte le informazioni di login vanno perdute. Il valore predefinito è 20. È possibile ridefinire il valore predefinito in qualsiasi sessione modificandone la proprietà *TimeoutMinutes*.

Pagine di login

Naturalmente, l’applicazione ha bisogno anche di una pagina di login. Gli utenti immettono il loro nome utente e la loro password per l’autenticazione, o mentre tentano di accedere a una pagina con accesso limitato o prima di tale tentativo. L’utente può anche specificare quale pagina ricevere una volta completata l’autenticazione. Se il nome utente e la password sono uguali a quelli di un utente nell’elenco Web user, l’utente acquisisce i diritti di accesso appropriati e viene indirizzato alla pagina specificata nella pagina di login. Se l’utente non è autenticato,

è possibile visualizzare di nuovo la pagina di login (azione predefinita) oppure fare eseguire un'altra azione.

Fortunatamente, WebSnap semplifica la creazione di una semplice pagina di login utilizzando un modulo pagina Web e l'adapter produttore di pagine. Per creare una pagina di login, iniziare creando un nuovo modulo pagina Web. Selezionare File | New | Other per visualizzare la finestra di dialogo New Items, quindi selezionare dalla pagina WebSnap l'opzione Page Module. Selezionare AdapterPageProducer come tipo di produttore di pagina. Compilare le altre opzioni nel modo desiderato. Come nome della pagina di login si può usare Login.

A questo punto si dovrebbero aggiungere i classici campi presenti in una pagina di login: un campo per il nome utente, uno per la password, una casella di selezione per selezionare la pagina da ricevere dopo aver effettuato il login e un pulsante Login che trasmette la pagina ed esegue l'autenticazione dell'utente. Per aggiungere questi campi:

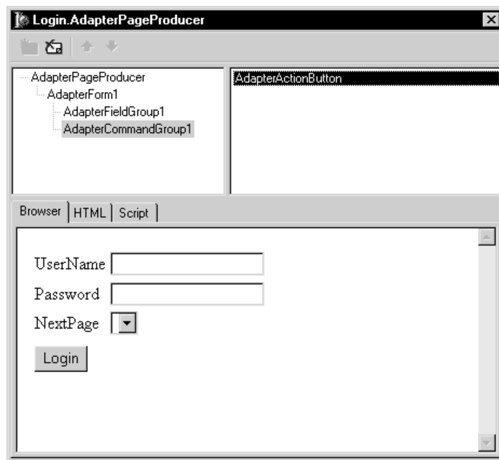
1. Aggiungere al modulo pagina Web appena creato un componente LoginFormAdapter (che è possibile trovare sulla pagina WebSnap della Component palette).
2. Fare doppio clic sul componente *AdapterPageProducer* per visualizzare una finestra per l'editing della pagina web.
3. Fare clic destro su *AdapterPageProducer* nel riquadro superiore di sisnistra e selezionare New Component. Nella finestra di dialogo Add Web Component selezionare *AdapterForm* e fare clic su OK.
4. Aggiungere un *AdapterFieldGroup* ad AdapterForm. (Fare clic destro su *AdapterForm* nel riquadro superiore di sisnistra e selezionare New Component. Nella finestra di dialogo Add Web Component selezionare *AdapterFieldGroup* e fare clic su OK.)
5. Passare all'Object Inspector e impostare la proprietà *Adapter* di *AdapterFieldGroup* a *LoginFormAdapter*. I campi UserName, Password e NextPage dovrebbero venire visualizzati automaticamente nella pagina Browser dell'editor della pagina web.

Fatto ciò, WebSnap si incarica della maggior parte del lavoro con poche semplici operazioni. Sulla pagina di login manaca ancora un pulsante Login che trasmetta le informazioni sulla scheda per l'autenticazione. Per aggiungere un pulsante Login:

1. Aggiungere un *AdapterCommandGroup* ad *AdapterForm*.
2. Aggiungere un *AdapterActionButton* ad *AdapterCommandGroup*.
3. Fare clic su *AdapterActionButton* (elencato nel riquadro superiore di destra dell'editor della pagina web) e, utilizzando l'Object Inspector, modificarne la proprietà *ActionName* in *Login*. È possibile vedere un'anteprima della pagina di login nella pagina Browser dell'editor della pagina web.

L'editor della pagina web dovrebbe sembrare assomigliare a quello visualizzato di seguito.

Figura 34.11 Un esempio di pagina di login visto nell'editor della pagina web



Se sotto *AdapterFieldGroup* non viene visualizzato il pulsante, accertarsi che *AdapterCommandGroup* sia elencato dopo *AdapterFieldGroup* nell'editor della pagina web. Se è visualizzato prima, selezionare *AdapterCommandGroup* e fare clic sulla freccia verso il basso sull'editor della pagina web. (Di solito, gli elementi delle pagine web vengono visualizzati in verticale nello stesso ordine con cui sono visualizzati nell'editor della pagina web.)

Prima che la pagina di login diventi funzionale è necessaria un'altra operazione. È necessario specificare qual è la pagina di login nell'adapter della sessione per l'utente finale. Per fare ciò, selezionare il componente *EndUserSessionAdapter* nel modulo Web application. Nell'Object Inspector, modificare la proprietà *LoginPage* nel nome della pagina di login. Ora la pagina di login è abilitata per tutte le pagine nell'applicazione per Web server.

Impostazione delle pagine in modo che richiedano il login

Una volta che si ha una pagina di login funzionante, è necessario che le pagine a cui è necessario accedere in modo controllato richiedano il login. Il modo più semplice per fare in modo che una pagina richieda il login consiste nel progettare tale pagina con quel requisito. All'atto delle creazione del modulo pagina Web, nella sezione Page della finestra di dialogo New WebSnap Page Module, spuntare la casella Login Required.

Se si crea una pagina senza richiedere il login, è possibile modificarla successivamente. Per fare in modo che venga richiesto il login dopo aver creato un modulo pagina Web:

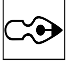
1 Aprire nell'editor il codice sorgente del file associato al modulo pagina Web.

2 Scorrere il sorgente fino alla dichiarazione della funzione statica WebInit.

3 Eliminare il commento dalla parte *wpLoginRequired* dell'elenco di parametri rimuovendo i simboli /* e */ che lo racchiudono. La funzione WebInit dovrebbe somigliare a quello visualizzata di seguito:

```
static TWebPageInit WebInit(__classid(TAdapterPageProducerPage3), crOnDemand, caCache,
PageAccess << wpPublished << wpLoginRequired, ".html", "", "", "", "");
```

Per eliminare da una pagina la necessità di effettuare il login, invertire il processo e commentare di nuovo la sezione *wpLoginRequired* della dichiarazione.



È possibile utilizzare lo stesso procedimento per rendere pubblica la pagina o meno. È sufficiente aggiungere o rimuovere i segni di commento che racchiudono la sezione *wpPublished* in base alle proprie necessità.

Diritti di accesso degli utenti

I diritti di accesso degli utenti sono una parte importante di qualsiasi applicazione per Web server. È necessario essere in grado di controllare chi può visualizzare e modificare le informazioni che il server fornisce. Ad esempio, si supponga di creare un'applicazione server per gestire le vendite al dettaglio online. È normale consentire agli utenti di visualizzare gli elementi nel catalogo, ma non si deve consentire che siano in grado di modificare i prezzi! Chiaramente, i diritti di accesso sono un argomento importante.

Per fortuna, WebSnap consente di controllare l'accesso alle pagine e ai contenuti del server in numerosi modi. Nelle sezioni precedenti, si è visto come è possibile controllare accesso alle pagine tramite la richiesta di login. Vi sono anche altre possibilità. Ad esempio:

- È possibile visualizzare i campi dati in una casella di modifica agli utenti con gli appropriati diritti di modifica; altri utenti vedranno il contenuto di campo, ma non avranno la possibilità di modificarli.
- È possibile nascondere determinati campi agli utenti che non hanno i necessari diritti di accesso in visualizzazione.
- È possibile impedire agli utenti non autorizzati di ricevere determinate pagine.

In questa sezione viene spiegato come implementare questi comportamenti.

Visualizzazione dinamica di campi come caselle di testo o di modifica

Se si utilizza l'adapter produttore di pagine, è possibile modificare l'aspetto degli elementi della pagina per gli utenti con diversi diritti di accesso. Ad esempio, la demo Biolife (presente nella subdirectory WebSnap della directory Demos) contiene una pagina di scheda che mostra tutte le informazioni per una certa specie. La scheda viene visualizzata quando l'utente fa clic su un pulsante Details sulla griglia. Un utente connesso come Will vede i dati presentati come testo. A Will non è consentito modificare i dati, e pertanto la scheda non gli offre alcun meccanismo per farlo. L'utente Ellen ha i diritti di modifica, e pertanto, quando Ellen visualizza la pagina della scheda, vede una serie di caselle di modifica che le permettono di modificare il contenuto dei campi. Un tale utilizzo dei diritti di accesso può evitare di creare ulteriori pagine.

L'aspetto di alcuni elementi della pagina, come *TAdapterDisplayField* e *TAdapterDisplayColumn*, è determinato dalla proprietà *ViewMode*. Se *ViewMode* è impostato a *vmToggleOnAccess*, l'elemento della pagina sarà visualizzato agli utenti con diritti di modifica come una casella di modifica. Gli utenti senza diritti di modifica vedranno solo il testo. Impostare la proprietà *ViewMode* a *vmToggleOnAccess* per fare in modo che l'aspetto e la funzione dell'elemento della pagina venga stabilito dinamicamente.

Un elenco di utenti Web è un elenco di oggetti *TWebUserListItem*, uno per ogni utente che può effettuare il login al sistema. Le autorizzazioni agli utenti sono memorizzate nella proprietà *AccessRights* del rispettivo elemento dell'elenco di utenti Web. *AccessRights* è una stringa di testo, pertanto si è liberi di specificare le autorizzazioni nel modo preferito. Creare un nome per ogni tipo di diritto di accesso desiderato all'applicazione server. Se si vuole che un utente abbia più diritti di accesso, separare gli elementi nell'elenco con uno spazio, con un punto e virgola o con una virgola.

I diritti di accesso per i campi sono controllati dalle ripettive proprietà *ViewAccess* e *ModifyAccess*. *ViewAccess* memorizza il nome dei diritti di accesso necessari per visualizzare un certo campo. *ModifyAccess* impone i diritti di accesso necessari per modificare i dati di campo. Queste proprietà appaiono in due punti: in ogni campo e nell'oggetto adapter che li contiene.

Il controllo dei diritti di accesso è un processo in due fasi. Nel momento in cui deve decidere l'aspetto di un campo in una pagina, l'applicazione per prima cosa controlla i diritti di accesso del campo. Se il valore è una stringa vuota, allora l'applicazione controlla i diritti di accesso dell'adapter che contiene il campo. Se anche la proprietà dell'adapter è vuota, l'applicazione seguirà il suo comportamento predefinito. Per gli accessi in modifica, il comportamento predefinito è quello di consentire le modifiche a tutti gli utenti nell'elenco utenti Web la cui proprietà *AccessRights* non è vuota. Per gli accessi in visualizzazione, l'autorizzazione viene concessa automaticamente se non è stato specificato alcun diritto di accesso in visualizzazione.

Occultamento dei campi e del loro contenuto

È possibile nascondere il contenuto di un campo agli utenti che non hanno gli opportuni diritti di visualizzazione. Per prima cosa si imposta la proprietà *ViewAccess* del campo in modo che corrisponda all'autorizzazione che si desidera assegnare agli utenti. Quindi, ci si deve accertare che il *ViewAccess* per l'elemento della pagina del campo sia impostato a *vmToggleOnAccess*. L'intestazione di campo sarà visualizzata, ma il valore del campo non lo sarà.

Naturalmente, spesso è meglio nascondere tutti i riferimenti al campo quando un utente non ha le autorizzazioni di visualizzazione. Per fare ciò, impostare *HideOptions* dell'elemento di pagina del campo in modo da includere *hoHideOnNoDisplayAccess*. Non saranno visualizzati né l'intestazione né i contenuti del campo.

Come evitare l'accesso alla pagina

Si potrebbe decidere che alcune pagine non devono essere accessibili agli utenti non autorizzati. Per concedere i diritti di accesso prima di visualizzare le pagine,

modificare la dichiarazione della funzione WebInit del modulo. Questa funzione appare nel codice sorgente del modulo.

La funzione WebInit accetta fino a 9 argomenti. Di solito WebSnap ne lascia quattro impostati ai valori predefiniti (stringhe vuote), pertanto di solito la chiamata assomiglia alla seguente:

```
static TWebPageInit WebInit(__classid(TAdapterPageProducerPage3), crOnDemand, caCache,
PageAccess << wpPublished /* << wpLoginRequired */, ".html", "", "", "", "");
```

Per controllare le autorizzazioni prima di consentire l'accesso, è necessario fornire nel nono parametro la stringa per la necessaria autorizzazione. Ad esempio, si supponga che il nome dell'autorizzazione sia Access. Ecco come si dovrebbe modificare la funzione WebInit:

```
static TWebPageInit WebInit(__classid(TAdapterPageProducerPage3), crOnDemand, caCache,
PageAccess << wpPublished /* << wpLoginRequired */, ".html", "", "", "", "Access");
```

L'accesso alla pagina sarà negato a chiunque è privo dell'autorizzazione Access.

Scripting lato server in WebSnap

I modelli dei produttori di pagine possono includere linguaggi di scripting come JScript o VBScript. Il produttore di pagine esegue lo script in risposta a una richiesta di contenuti del produttore. Poiché l'applicazione per Web server valuta lo script, viene detta script lato server, a differenza dello script lato client (che viene valutato dal browser).

Questa sezione fornisce una panoramica concettuale dello scripting lato server e di come viene utilizzato dalle applicazioni WebSnap. L'appendice "Guida di riferimento allo scripting lato server di WebSnap" contiene informazioni molto più dettagliate sugli oggetti script e sulle rispettive proprietà e metodi. La si può considerare come una guida di riferimento alle API per lo scripting lato server, simile alle descrizioni degli oggetto di C++Builder incluse nei file della Guida. L'appendice contiene anche esempi dettagliati di script che mostrano esattamente come si può utilizzare lo script per generare pagine HTML.

Benché lo scripting lato server sia una parte importante di WebSnap, non è necessario utilizzare gli script nelle applicazioni WebSnap. Lo scrip viene utilizzato per la generazione del codice HTML e per null'altro. Permette di inserire in una pagina HTML i dati dell'applicazione. Infatti, quasi tutte le proprietà esposte dagli adattatori e altri oggetti abilitati allo script sono a sola lettura. Lo script lato server non viene utilizzato per modificare i dati dell'applicazione, che vengono gestiti da componenti e da gestori di evento scritti in Pascal o in C++.

Esistono altri modi per inserire i dati dell'applicazione in una pagina HTML. Se si preferisce, è possibile utilizzare i marcatori trasparenti di Web Broker o qualche altra soluzione basata su marcatori. Ad esempio, diversi progetti nella cartella Examples\WebSnap di C++ Builder utilizzano XML e XSL invece dello scripting. Senza scripting, tuttavia, si sarà costretti a scrivere la maggior parte della logica della generazione HTML in C++, il che allungherà i tempi di sviluppo.

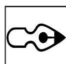
Lo scripting utilizzato in WebSnap è orientato agli oggetti e supporta la logica condizionale e i cicli, il che può semplificare notevolmente l’attività di generazione delle pagine. Ad esempio, le pagine possono includere un campo dati che può essere modificato da alcuni utenti ma non da altri. Con lo scripting, la logica condizionale può essere inclusa nelle pagine modello che visualizzeranno una casella di modifica per gli utenti autorizzati e del semplice testo per altri. Con un approccio basato su marcatori, è necessario programmare tale processo decisionale nel codice sorgente che genera il codice HTML.

Active scripting

Per implementare lo script lato server, WebSnap si basa su *active scripting*. Active scripting è una tecnologia creata da Microsoft per consentire l’uso di un linguaggio di scripting con gli oggetti dell’applicazione attaraverso interfacce COM. Microsoft prevede due linguaggi di active scripting, VBScript e JScript. Il supporto per altri linguaggi è disponibile tramite produttori terzi.

Motore di script

La proprietà *ScriptEngine* del produttore di pagine identifica il motore di active scripting che valuta lo script all’interno di un modello. L’impostazione predefinita è quella di supportare JScript, ma può anche supportare altri linguaggi di scripting (come VBScript).



Gli adattatori di WebSnap sono progettati per produrre JScript. Per altri linguaggi di scripting sarà necessario fornire allo script la logica di generazione.

Blocchi di script

I blocchi di script, che appaiono nei modelli HTML, sono delimitati da <% e da %>. Il motore di script valuta qualsiasi testo interno al blocco di script. Il risultato diventa parte dei contenuti del produttore di pagine. Il produttore di pagine scrive il testo esterno a un blocco di script dopo avere tradotto gli eventuali marcatori trasparenti incorporati. I blocchi di script possono anche includere testo, consentendo alla logica condizionale e ai cicli di determinare il testo di output. Ad esempio, il seguente blocco JScript genera un elenco di cinque righe numerate:

```
<ul>
<% for (i=0;i<5;i++) { %>
    <li>Item <%=i %></li>
<% } %>
</ul>
```

(TIl delimitatore <%= è l’abbreviazione di *Response.Write*)

Creazione di script

Gli sviluppatori possono sfruttare le funzionalità di WebSnap per generare automaticamente script.

Modelli di wizard

Quando si crea un nuovo modulo WebSnap application o pagina, i wizard di WebSnap presentano un campo Template che viene utilizzato per selezionare il contenuto iniziale del modello del modulo pagina. Ad esempio, il modello Default genera JScript che, a sua volta, visualizza il titolo dell'applicazione, il nome della pagina e i link alle pagine pubblicate.

TAdapterPageProducer

TAdapterPageProducer crea schede e tabelle mediante la generazione di HTML e JScript. Lo JScript generato chiama gli oggetti adapter per acquisire i valori dei campi, parametri dei campi immagine e i parametri delle azioni.

Modifica e visualizzazione di script

Per visualizzare l'HTML risulatante dall'esecuzione dello script utilizzare la pagina HTML Result. Utilizzare la pagina Preview per visualizzare il risultato in un browser. La pagina HTML Script è disponibile se il modulo pagina Web utilizza *TAdapterPageProducer*. La pagina HTML Script visualizza il codice HTML e JScript generati dall'oggetto *TAdapterPageProducer*. Consultare questa vista per vedere come scrivere script che costruiscono schede HTML per visualizzare campi dell'adapter ed eseguire azioni dell'adapter.

Inclusione di script in una pagina

Un modello può includere script provenienti da un file o da un'altra pagina. Per includere script provenieenti da un file, utilizzare la seguente istruzione:

```
<!-- #include file="filename.html" -->
```

Se il modello include script di un'altra pagina, lo script viene valutato dalla pagina che effettua l'inclusione. Per includere il contenuto non valutato di page1, utilizzare la seguente istruzione.

```
<!-- #include page="page1" -- >
```

Oggetti script

Gli oggetti script sono oggetti che i comandi dello script possono referenziare. È possibile rendere gli oggetti disponibili allo scripting registrando un'interfaccia *IDispatch* verso l'oggetto con il motore di active scripting. I seguenti oggetti sono

disponibili per lo scripting:

Tabella 34.4 Oggetti script

Oggetto script	Descrizione
Application	Consente l'accesso all'adapter applicazione del modulo Web Application.
EndUser	Consente l'accesso all'adapter utente finale del modulo Web Application.
Session	Consente l'accesso all'oggetto sessione del modulo Web Application.
Pages	Consente l'accesso alle pagine dell'applicazione.
Modules	Consente l'accesso ai moduli dell'applicazione.
Page	Consente l'accesso alla pagina corrente
Producer	Consente l'accesso al produttore di pagina del modulo pagina Web.
Response	Consente l'accesso a WebResponse. Utilizzare questo oggetto quando non si desidera la sostituzione dei marcatori.
Request	Consente l'accesso a WebRequest.
Adapter objects	Tutti i componenti adapter sulla pagina corrente possono essere referenziati senza qualifica. Gli adattatori in altri moduli devono essere qualificati utilizzando gli oggetti Modules.

Gli oggetti script sulla pagina corrente, che utilizzano tutti lo stesso adapter, possono essere referenziati senza qualifica. Gli oggetti script su altre pagine fanno parte di un altro modulo pagina e hanno un oggetto adapter diverso. È possibile accedervi anteponendo al riferimento dell'oggetto script il nome dell'oggetto adapter. Ad esempio,

```
<%= FirstName %>
```

visualizza il contenuto della proprietà *FirstName* dell'adapter della pagina corrente. La seguente riga di script visualizza la proprietà *FirstName* di Adapter1, che è in un altro modulo di pagina:

```
<%= Adapter1.FirstName %>
```

Per una descrizione esaustiva degli oggetti script, vedere l'appendice "Guida di riferimento per lo script lto server di WebSnap.

Indirizzamento di richieste e risposte

Un motivo per utilizzare WebSnap per lo sviluppo di applicazioni per Web server è che componenti WebSnap gestiscono automaticamente le richieste e le risposte HTML. Invece di scrivere gestori di evento per i soliti compiti gravosi di trasferimento delle pagine, è possibile incentrare i propri sforzi sulla logica di gestione e sulla progettazione del server. Nonostante ciò, può essere utile capire in che modo le applicazioni WebSnap gestiscono le richieste e le risposte HTML. Questa sezione presenta una panoramica di tale processo.

Prima di gestire qualsiasi richiesta, il modulo Web application inizializza l'oggetto (di tipo *TWebContext*) Web context. L'oggetto Web context, a cui si accede chiamando la funzione globale WebContext, fornisce l'accesso globale alle variabili utilizzate dai componenti che soddisfano la richiesta. Ad esempio, Web context contiene gli oggetti

TWebRequest e *TWebResponse* per rappresentare il messaggio di richiesta HTTP e la risposta che dovrebbero essere restituiti.

Componenti dispatcher

I componenti dispatcher nel modulo Web application controllano il flusso dell'applicazione. I dispatcher determinano come gestire alcuni tipi di messaggi di richiesta HTTP esaminando la richiesta HTTP.

Il componente adapter dispatcher (*TAdapterDispatcher*) cerca un campo di contenuti o un campo query, che identifica un componente adapter action o un componente adapter image field. Se l'adapter dispatcher trova un componente, passa controllo a quel componente.

Il componente Web dispatcher (*TWebDispatcher*) gestisce una raccolta di elementi di azione (di tipo *TWebActionItem*) che sanno come gestire alcuni tipi di messaggi di richiesta HTTP. Il Web dispatcher cerca un elemento di azione che soddisfi la richiesta. Se ne trova uno, passa il controllo a quell'elemento di azione. Il Web dispatcher cerca anche componenti auto-dispatching che possano gestire la richiesta.

Il componente page dispatcher (*TPageDispatcher*) esamina la proprietà PathInfo dell'oggetto *TWebRequest* , cercando il nome di un modulo pagina Web registrato. Se il dispatcher trova un nome di modulo pagina Web, passa il controllo a quel modulo.

Operazioni dell'adapter dispatcher

Il componente adapter dispatcher (*TAdapterDispatcher*) gestisce automaticamente l'inoltro di schede HTML e le richieste di immagini dinamiche, chiamando i componenti adapter action e field.

Utilizzo di componenti adapter per generare contenuti

Affinché le applicazioni WebSnap eseguano automaticamente le azioni dell'adapter e ottengano le immagini dinamiche dai campi dell'adapter, è necessario che il contenuto HTML sia costruito correttamente. Se il contenuto HTML non è costruito correttamente, la richiesta HTTP risultante non conterrà le informazioni di cui necessita il dispatcher degli adapter per chiamare i componenti adapter action e campo.

Per ridurre gli errori nella costruzione della pagina HTML, i componenti adapter indicano i nomi e i valori degli elementi HTML. I componenti adapter hanno metodi che acquisiscono i nomi e i valori di campi nascosti che devono apparire su una scheda HTML progettata per aggiornare i campi dell'adapter. Di solito, i produttori di pagine utilizzano lo script lato server per acquisire i nomi e i valori dai componenti adapter e quindi utilizzano queste informazioni per generare il codice HTML. Ad esempio, il seguente script costruisce un elemento <IMG> che referenzia il campo di nome Graphic di Adapter1:

```
<img src="<%=Adapter1.Graphic.Image.AsHREF%>" alt="<%=Adapter1.Graphic.DisplayText%>">
```

Quando l'applicazione web valuta lo script, l'attributo HTML src conterrà le informazioni necessarie per identificare il campo e tutti i parametri necessari al componente campo per acquisire l'immagine. Il codice HTML risultante potrebbe assomigliare a questo:

```
<img src="?_lSpecies No=90090&__id=DM.Adapter1.Graphic" alt="(GRAPHIC)">
```

Quando il browser manda una richiesta HTTP per acquisire questa immagine all'applicazione web, il dispatcher degli adapter sarà in grado di determinare che il campo Graphic di Adapter1, nel modulo DM, dovrebbe essere chiamato con il parametro "Species No=90090". Il dispatcher degli adapter chiamerà il campo Graphic per scrivere una risposta HTTP appropriata.

Il seguente script costruisce un elemento <A> che referenzia l'azione EditRow di Adapter1 e crea un collegamento ipertestuale a una pagina chiamata Details:

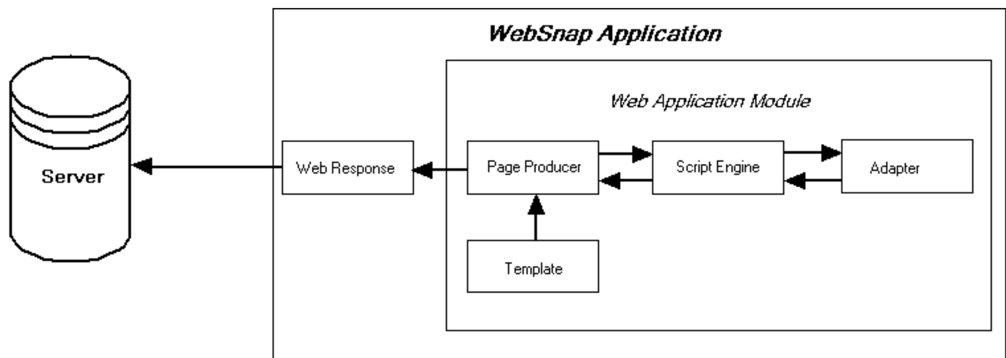
```
<a href="<%=Adapter1.EditRow.LinkToPage("Details", Page.Name).AsHREF%>">Edit...</a>
```

Il codice HTML risultante potrebbe assomigliare a questo:

```
<a href="?&_lSpecies No=90310&__sp=Edit&__fp=Grid&__id=DM.Adapter1.EditRow">Edit...</a>
```

L'utente finale fa clic su questo collegamento ipertestuale e il browser invia una richiesta HTTP. Il dispatcher degli adapter è in grado di determinare che l'azione EditRow di Adapter1, nel modulo DM, dovrebbe essere chiamata con il parametro Species No=903010. Il dispatcher degli adapter visualizza la pagina Edit se l'azione va a buon fine e visualizza la pagina Grid se l'esecuzione dell'azione non riesce. Quindi chiama l'azione EditRow per individuare la riga da modificare e viene chiamata la pagina di nome Edit per generare una risposta HTTP. La Figura 34.12 mostra come sono utilizzati i componenti adapter per generare dei contenuti.

Figura 34.12 Generazione del flusso dei contenuti



Ricezione di richieste dell'adapter e generazione delle risposte

Quando il dispatcher degli adattatori riceve una richiesta client, il dispatcher degli adattatori crea oggetti adapter request e adapter response per conservare le informazioni relative a quella richiesta HTTP. Gli oggetti oggetti adapter request e adapter response sono memorizzati nel Web context per consentire l'accesso durante l'elaborazione della richiesta.

Il dispatcher degli adapter crea due tipi di oggetti adapter request: action e image. Crea l'oggetto richiesta di azione quando esegue un'azione di adapter. Crea l'oggetto richiesta di immagine quando ricerca un'immagine da un campo dell'adapter.

L'oggetto adapter response è utilizzato dal componente adapter per indicare la risposta a una richiesta adapter action o adapter image. Esistono due tipi di oggetti adapter response, action e image.

Richieste di azioni

Gli oggetti di richiesta di azioni sono responsabili di suddividere la richiesta HTTP in informazioni necessarie per eseguire un'azione dell'adapter. I tipi di informazioni necessarie per l'esecuzione di un'azione dell'adapter possono includere le seguenti informazioni di richiesta:

Tabella 34.5 Informazioni di richiesta presenti nelle richieste di azioni

Informazione della richiesta	Descrizione
Nome componente	Identifica il componente adapter action.
Adapter mode	Definisce una modalità. Ad esempio, *TDataSetAdapter* supporta le modalità Edit, Insert e Browse. Un adapter action può comportarsi in diversi modi a seconda della modalità.
Success page	Identifica la pagina visualizzata dopo l'esecuzione con successo dell'azione.
Failure page	Identifica la pagina visualizzata nel caso si verifiche un errore durante l'esecuzione dell'azione.
Action request parameters	Identifica i parametri necessari all'adapter action. Ad esempio, l'azione Apply di *TdataSetAdapter* includerà i valori chiave che che identificano il record da aggiornare.
Adapter field values	Specifica valori dei campi dell'adapter passati nella richiesta HTTP quando si trasmette una scheda HTML. Un valore del campo può includere nuovi valori immessi dall'utente finale, i valori originali del campo dell'adapter e i file caricati.
Record keys	Specifica le chiavi che identificano in modo univoco ogni record.

Generazione delle risposte di azione

Gli oggetti action response generano una risposta HTTP per conto di un componente adapter action. L'adapter action indica il tipo di risposta impostando proprietà all'interno dell'oggetto o chiamando metodi nell'oggetto action response. Le proprietà includono:

- *RedirectOptions*—Le opzioni di reindirizzamento indicano se eseguire un reindirizzamento HTTP invece di restituire dei contenuti HTML.
- *ExecutionStatus*—L'impostazione dello stato a Success fa in modo che l'action response predefinita sia il contenuto della pagina Success identificata nell'Action Request.

I metodi di action response includono:

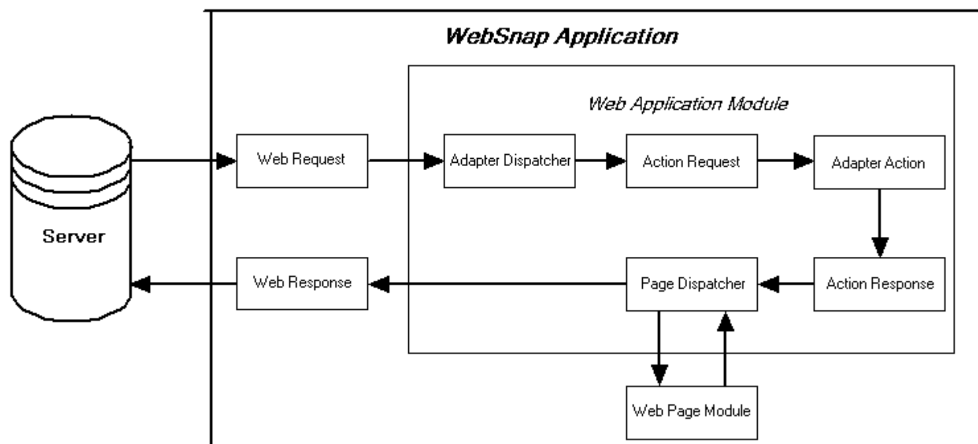
- *RespondWithPage* —L'adapter action chiama questo metodo quando un particolare modulo pagina Web dovrebbe generare la risposta.

- *RespondWithComponent*—L'adapter action chiama questo metodo quando la risposta dovrebbe venire dal modulo pagina Web che contiene questo componente.
- *RespondWithURL*—L'adapter action chiama questo metodo quando la risposta è un reindirizzamento all'URL specificato.

Quando risponde con una pagina, l'oggetto action response tenta di utilizzare il dispatcher delle pagine per generare contenuti della pagina. Se non trova il dispatcher delle pagine, chiama direttamente il modulo pagina Web.

La Figura 34.15 illustra il modo in cui gli oggetti action request and action response gestiscono una richiesta.

Figura 34.13 Action request e response



Richiesta di immagine

informazioni richieste dal campo immagine dell'adapter per generare un'immagine. I tipi di informazioni rappresentate da Image Request includono:

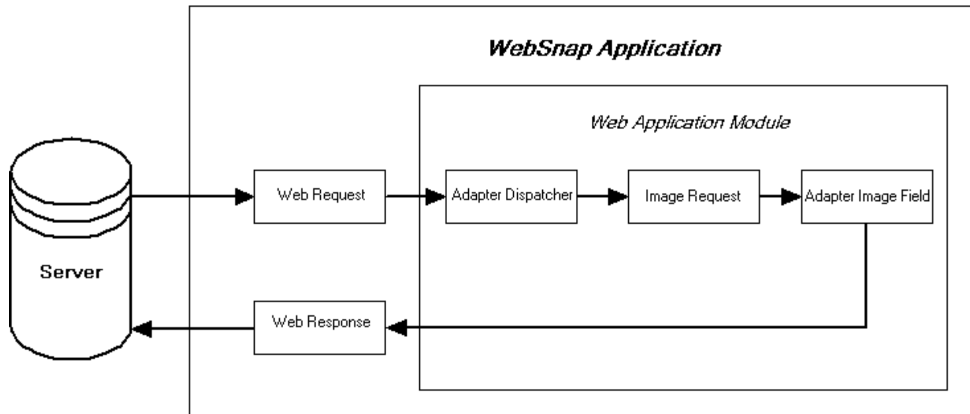
- Component name - Identifica il componente campo dell'adapter.
- Image request parameters - Identifica i parametri necessari per l'adapter image. Ad esempio, all'oggetto *TDataSetAdapterImageField* sono necessari dei valori chiave per identificare il record che contiene l'immagine.

Risposta di immagine

L'oggetto image response contiene l'oggetto *TWebResponse*. I campi dell'adapter rispondono a una richiesta dell'adapter scrivendo un'immagine nell'oggetto Web response.

La Figura 34.14 illustra illustra il modo in cui i campi immagine dell'adapter rispondono a una richiesta.

Figura 34.14 Risposta di immagine a una richiesta



Indirizzamento degli elementi di azione

Quando risponde a una richiesta, il Web dispatcher (*TWebDispatcher*) ricerca nel suo elenco di elementi di azione quello che:

- corrisponde alla parte PathInfo dell'URL di destinazione del messaggio di richiesta, e
- è in grado di fornire il servizio specificato come il metodo del messaggio di richiesta.

L'operazione viene svolta confrontando le proprietà *PathInfo* e *MethodType* dell'oggetto *TWebRequest* con le proprietà aventi lo stesso nome sull'elemento di azione.

Quando il dispatcher trova l'elemento di azione appropriato, provoca l'innesco di quell'elemento di azione. Quando l'elemento di azione viene innescato, fa una delle seguenti cose:

- Compila il contenuto della risposta e invia la risposta o segnala che la richiesta è stata gestita completamente.
- Aggiunge alla risposta e quindi permette ad altri elementi di azione di completare il lavoro.
- Rimanda la richiesta ad altri elementi di azione.

Dopo che il dispatcher ha controllato tutti i suoi elementi di azione, se il messaggio non è stato gestito correttamente, il dispatcher controlla la presenza di componenti di auto-dispatching registrati appositamente che non utilizzano elementi di azione. (Questi componenti sono specifici per le applicazioni database multi-tiered). Se il messaggio di richiesta non viene ancora gestito completamente, il dispatcher chiama l'elemento di azione predefinito. Non è necessario che l'URL di destinazione o il metodo della richiesta corrispondano all'elemento di azione predefinito.

Operazione del dispatcher di pagine

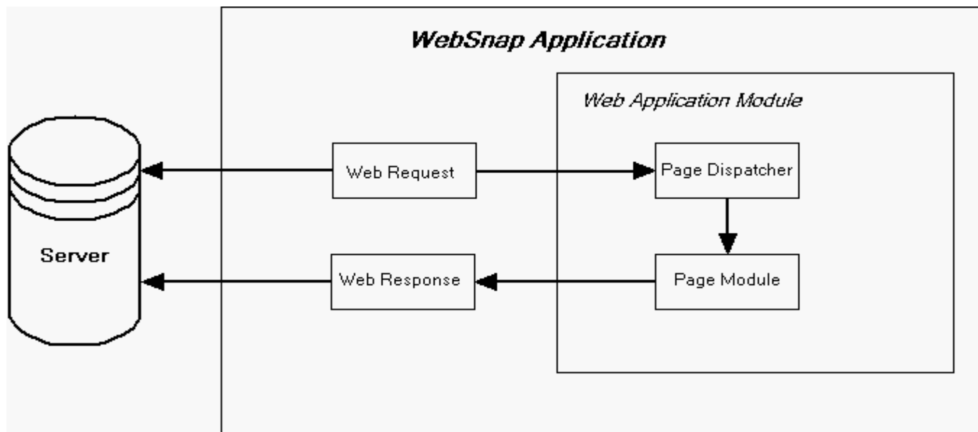
Quando il dispatcher di pagine riceve una richiesta del client, determina il nome della pagina controllando la parte PathInfo del messaggio di richiesta dell'URL di destinazione. Se la parte PathInfo non è vuota, il dispatcher di pagine utilizza come nome della pagina la parola finale di PathInfo. Se la parte PathInfo è vuota, il dispatcher di pagine cerca di determinare un nome di pagina predefinito.

Se la proprietà *DefaultPage* del dispatcher di pagine contiene un nome di pagina, il dispatcher di pagine utilizza questo nome come nome di pagina predefinito. Se la proprietà *DefaultPage* è vuota e il modulo Web application è un modulo pagina, il dispatcher di pagine utilizza il nome del modulo Web application come nome di pagina predefinito.

Se il nome della pagina non è vuoto, il dispatcher di pagine cerca un modulo pagina Web con un nome corrispondente. Se trova un modulo pagina Web, chiama quel modulo per generare una risposta. Se il nome della pagina è vuoto o se il dispatcher di pagine non trova un modulo pagina Web, il dispatcher di pagine solleva un'eccezione.

La Figura 34.15 mostra come il dispatcher delle pagine risponde a una richiesta.

Figura 34.15 Indirizzamento di una pagina



Capitolo

35

Operazioni con documenti XML

XML (Extensible Markup Language) è un linguaggio di markup per la descrizione di dati strutturati. È simile a HTML, tranne che i marcatori descrivono la struttura delle informazioni invece delle caratteristiche di visualizzazione. I documenti XML rappresentano un modo semplice, basato su testo, per memorizzare informazioni in modo da semplificarne la ricerca o la modifica. Essi sono utilizzati spesso come formato standard e trasportabile per dati in applicazioni Web, in comunicazioni business-to-business e così via.

I documenti XML forniscono una vista gerarchica di un corpo di dati. I marcatori nel documento XML descrivono il ruolo o il significato di ogni dato, come si vede nel seguente documento, che descrive una raccolta di titoli azionari:

```
<?xml version="1.0" encoding=UTF-8 standalone="no" ?>
<!DOCTYPE StockHoldings SYSTEM "sth.dtd">
<StockHoldings>
   <Stock exchange="NASDAQ">
      <name>Borland</name>
      <price>15.375</price>
      <symbol>BORL</symbol>
      <shares>100</shares>
   </Stock>
   <Stock exchange="NYSE">
      <name>Pfizer</name>
      <price>42.75</price>
      <symbol>PFE</symbol>
      <shares type="preferred">25</shares>
   </Stock>
</StockHoldings>
```

Questo esempio illustra molti elementi tipici in un documento XML. La prima riga è un'istruzione di elaborazione che viene chiamata dichiarazione di XML. La dichiarazione XML è facoltativa ma si dovrebbe includerla in quanto fornisce informazioni utili sul documento. In questo esempio, la dichiarazione XML dice che il documento è conforme alla versione 1.0 delle specifiche XML, che utilizza la

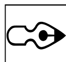
codifica UTF-8 e che si basa su un file esterno per la dichiarazione del tipo di documento (DTD)).

La seconda riga, che inizia con il marcatore <!DOCType>, è una dichiarazione del tipo di documento. La dichiarazione del tipo di documento (DTD) è il modo in cui XML definisce la struttura del documento. Essa impone regole sintattiche relative agli elementi (marcatori) contenuto nel documento. Il DTD in questo esempio referenzia un altro file (sth.dtd). In questo caso, la struttura è definita in un file esterno, invece che all'interno dello stesso documento XML. Altri tipi di file che descrivono la struttura di un documento XML includono XDR (Reduced XML Data) e XDR (XML schema).

Le restanti righe sono organizzate gerarchicamente con un singolo nodo radice (il marcatore <StockHoldings>). Ogni nodo in questa gerarchia contiene o un insieme di nodi figlio, o un valore di testo. Alcuni marcatori (i marcatori <Stock> e <shares>) includono attributi, che sono coppie Name=Value che forniscono dettagli su come interpretare il marcatore.

Benché sia possibile operare direttamente con il testo in un documento XML, di solito le applicazioni utilizzano particolari strumenti aggiuntivi per l'analisi sintattica e la modifica dei dati. W3C definisce un insieme di interfacce standard per rappresentare un documento analizzato XML di nome DOM (Document Object Model). Molti fornitori forniscono parser XML che implementano le interfacce DOM per semplificare l'interprezione e la modifica di documenti XML.

C++Builder fornisce molti strumenti aggiuntivi per operare con i documenti XML. Questi strumenti utilizzano un parser DOM, fornito da un altro produttore, e facilitano notevolmente le operazioni sui documenti XML. Questo capitolo descrive questi strumenti.



Oltre agli strumenti descritti in questo capitolo, C++Builder dispone di strumenti e componenti per la conversione di documenti XML in pacchetti dati che si integrano nell'architettura database di C++Builder. Per maggiori informazioni sugli strumenti per integrare i documenti XML nelle applicazioni database, consultare il Capitolo 30, "Uso di XML nelle applicazioni database".

Utilizzo del Document Object Model

Il Document Object Model (DOM) è un insieme di interfacce standard per rappresentare un documento XML analizzato. Per impostazione predefinita, nel prodotto è inclusa una copia dell'implementazione DOM di Microsoft. Inoltre, include un meccanismo di registrazione che permette di integrare nel framework XML ulteriori implementazioni DOM di altri produttori.

La unit XMLDOM include le dichiarazioni per tutte le interfacce DOM definite nelle specifiche W3C XML DOM level 2. Ogni produttore DOM fornisce una propria implementazione per queste interfacce.

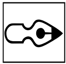
- Per utilizzare l'implementazione Microsoft, includere nel file sorgente l'header della unit MSXMLDOM. Poiché l'implementazione Microsoft è basata su COM, è

inoltre necessario registrare la libreria msxml.dll come server COM, qualora non lo fosse già. Per registrare questa DLL è possibile utilizzare Regsvr32.exe.

- Per utilizzare un'altra implementazione DOM, è necessario creare una unit che definisca un derivato della classe *TDOMVendor*. Nella classe discendente, è necessario ridefinire due metodi: il metodo *Description*, che restituisce una stringa che identifica il fornitore e il metodo *DOMImplementation*, che restituisce l'interfaccia di alto livello (*IDOMImplementation*). Si dovrebbe registrare questo fornitore chiamando la procedura globale *RegisterDOMVendor*. È possibile annullare la registrazione del fornitore chiamando la procedura globale *UnRegisterDOMVendor*. Una volta che si è registrato il nuovo DOMVendor, l'applicazione ha accesso all'implementazione DOM sottostante finché non si elimina la registrazione del fornitore.

Alcuni produttori forniscono estensioni alle interfacce DOM standard. Per consentire l'utilizzo di queste estensioni, la unit XMLDOM definisce anche un'interfaccia *IDOMNodeEx*. *IDOMNodeEx* è un discendente della *IDOMNode* standard che include la più utile di queste estensioni.

È possibile operare direttamente con interfacce DOM per analizzare e modificare documenti XML. È sufficiente chiamare la funzione *GetDOM* per ottenere un'interfaccia *IDOMImplementation*, che è possibile utilizzare come punto di partenza.



Per una descrizione dettagliata delle interfacce DOM, vedere le dichiarazioni nello header della unit XMLDOM, la documentazione fornita dal produttore DOM o le specifiche fornite sul sito Web di W3C (www.w3.org).

Potrebbe però risultare più conveniente utilizzare le classi XML invece di operare direttamente con le interfacce DOM. Queste classi sono descritte di seguito.

Operazioni con i componenti XML

La VCL (o la CLX) definisce molte classi e interfacce per operare con i documenti XML. Questi semplificano il processo di caricamento, modifica e salvataggio dei documenti XML.

Utilizzo di TXMLDocument

Il punto di partenza per operare con un documento XML è il componente *TXMLDocument*. I seguenti punti descrivono come utilizzare *TXMLDocument* per operare direttamente con un documento XML:

1. Aggiungere un componente *TXMLDocument* alla scheda o al modulo dati. *TXMLDocument* è presente sulla pagina Internet della Component palette.
2. Impostare la proprietà *DOMVendor* per specificare l'implementazione DOM che si vuole venga utilizzata dal componente per l'analisi sintattica e la modifica di un

documento XML. L'Object Inspector elenca tutti i produttori DOM attualmente registrati. Per informazioni sulle implementazioni DOM, consultare "Utilizzo del Document Object Model" a pagina 35-2.

3 A seconda dell'implementazione, si potrebbe voler impostare la proprietà *ParseOptions* in modo da configurare in che modo l'implementazione DOM sottostante analizza il documento XML.

4 Se si sta operando con un documento XML esistente, specificare il documento:

- Se il documento XML è memorizzato in un file, impostare la proprietà *FileName* al nome di quel file.
- Utilizzando invece la proprietà *XML* è possibile specificare il documento XML come una stringa.

5 Impostate la proprietà *Active* a **true**.

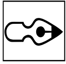
Una volta che si ha un oggetto *TXMLDocument* attivo, è possibile esaminare la gerarchia dei suoi nodi oltre che leggere o impostare i rispettivi valori. Il nodo radice di questa gerarchia è disponibile come proprietà *DocumentElement*.

Operazioni con i nodi XML

Una volta che un documento XML è stato analizzato da un'implementazione DOM, i dati che esso rappresenta sono disponibili sotto forma di gerarchia di nodi. Ogni nodo corrisponde a un elemento marcato nel documento. Ad esempio, dato il seguente XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE StockHoldings SYSTEM "sth.dtd">
<StockHoldings>
   <Stock exchange="NASDAQ">
      <name>Borland</name>
      <price>15.375</price>
      <symbol>BORL</symbol>
      <shares>100</shares>
   </Stock>
   <Stock exchange="NYSE">
      <name>Pfizer</name>
      <price>42.75</price>
      <symbol>PFE</symbol>
      <shares type="preferred">25</shares>
   </Stock>
</StockHoldings>
```

TXMLDocument genererebbe una gerarchia di nodi di questo tipo: La radice della gerarchia sarebbe il nodo *StockHoldings*. *StockHoldings* avrebbe due nodi figlio, che corrispondono alle due marcatori *Stock*. Ognuno di questi due nodi figlio avrebbe quattro nodi figlio propri (*name*, *price*, *symbol* e *shares*). Quei quattro nodi figlio agirebbero come nodi foglia. Il testo che essi contengono verrebbe visualizzato come il valore di ognuno dei nodi foglia.



Questa divisione in nodi è leggermente diversa dal modo in cui un'implementazione DOM genera i nodi per un documento XML. In particolare, un parser DOM tratta tutti gli elementi marcati come nodi interni. Per i valori dei *name*, *price*, *symbol* e *shares* verrebbero creati dei nodi aggiuntivi (di tipo nodo di testo). Questi nodi di testo sarebbero quindi visualizzati come figli dei nodi *name*, *price*, *symbol* e *shares*.

Ad ogni nodo si accede attraverso un'interfaccia *IXMLNode* a partire dal nodo radice, che è il valore della proprietà *DocumentElement*.

Operazioni con il valore di un nodo

Data un'interfaccia *IXMLNode*, è possibile controllare se rappresenta un nodo interno o un nodo foglia controllando la proprietà *IsTextElement*.

- Se rappresenta un nodo foglia, è possibile leggerne o impostarne il valore utilizzando la proprietà *Text*.
- Se rappresenta un nodo interno, è possibile accedere ai suoi nodi figlio utilizzando la proprietà *ChildNodes*.

Così, ad esempio, utilizzando il documento XML visto in precedeza, è possibile leggere il prezzo dell'azione Borland nel seguente modo:

```
_di_IXMLNode BorlandStock = XMLDocument1->DocumentElement->ChildNodes->GetNode(0);
AnsiString Price = BorlandStock->ChildNodes->Nodes[WideString("price")]->Text;
```

Operazioni con gli attributi di un nodo

Se il nodo include eventuali attributi, è possibile operare su di essi utilizzando la proprietà *Attributes*. È possibile leggere o modificare il valore di un attributo specificando il nome di un attributo esistente. Quando si imposta la proprietà *Attributes* è possibile aggiungere nuovi attributi specificando un nuovo nome di attributo:

```
_di_IXMLNode BorlandStock = XMLDocument1->DocumentElement->ChildNodes->GetNode(0);
BorlandStock->ChildNodes->Nodes[WideString("shares")]->Attributes[WideString("type")] =
"common";
```

Aggiunta e cancellazione di nodi figlio

È possibile aggiungere nodi figlio utilizzando il metodo *AddChild*. *AddChild* crea nuovi nodi che corrispondono a elementi contrassegnati nel documento XML. Tali nodi sono chiamati nodi elemento.

Per creare un nuovo nodo elemento, specificare il nome che apparirà nel nuovo marcatore e, volendo, la posizione in cui dovrebbe essere visualizzato il nuovo nodo. Ad esempio, il seguente codice aggiunge un nuovo titolo azionario al documento visto in precedenza:

```
_di_IXMLNode NewStock = XMLDocument1->DocumentElement->AddChild(WideString("stock"));
NewStock->Attributes[WideString("exchange")] = "NASDAQ";
_di_IXMLNode ValueNode = NewStock->AddChild(WideString("name"));
ValueNode->Text = WideString("Cisco Systems");
ValueNode = NewStock->AddChild(WideString("price"));
ValueNode->Text = WideString("62.375");
ValueNode = NewStock->AddChild(WideString("symbol"));
```

```
ValueNode->Text = WideString("CSCO");
ValueNode = NewStock->AddChild(WideString("shares"));
ValueNode->Text = WideString("25");
```

Una versione ridefinita di *AddChild* permette di specificare il namespace URI in cui è definito il nome del marcatore.

È possibile cancellare i nodi figlio utilizzando i metodi della proprietà *ChildNodes*. *ChildNodes* è un'interfaccia *IXMLNodeList*, che gestisce i figli di un nodo. È possibile utilizzare il metodo *Delete* per cancellare un singolo nodo figlio identificato in base alla posizione o al nome. Ad esempio, il seguente codice cancella l'ultimo titolo azionario nel documento visto in precedenza:

```
_di_IXMLNode StockList = XMLDocument1->DocumentElement;
StockList->ChildNodes->Delete(StockList->ChildNodes->Count - 1);
```

Astrazione di documenti XML con Data Binding wizard

Benché sia possibile operare con un documento XML utilizzando solo il componente *TXMLDocument* e l'interfaccia *IXMLNode*, che fa emergere i nodi in quel documento, o, addirittura, operare esclusivamente con le interfacce DOM (evitando anche *TXMLDocument*), mediante l'utilizzo di XML Data Binding wizard è possibile scrivere del codice molto più semplice e più facilmente leggibile.

Data Binding wizard prende uno schema XML o un file di dati e genera un set di interfacce che ne tracciano una mappa. (Un'interfaccia è una classe che contiene solo membri *pure virtual*). Si supponga di avere, ad esempio, una serie di dati XML simile alla seguente:

```
<customer id=1>
  <name>Mark</name>
  <phone>(831) 431-1000</phone>
</customer>
```

Data Binding wizard genera la seguente interfaccia (oltre a una classe per implementarla):

```
__interface INTERFACE_UUID("{F3729105-3DD0-1234-80e0-22A04FE7B451}") ICustomer :
    public IXMLNode
{
public:
  virtual int __fastcall  Getid(void) = 0 ;
  virtual DOMString __fastcall Getname(void) = 0 ;
  virtual DOMString __fastcall Getphone(void) = 0 ;
  virtual void __fastcall  Setid(int Value)= 0 ;
  virtual void __fastcall  Setname(DOMString Value)= 0 ;
  virtual void __fastcall  Setphone(DOMString Value)= 0 ;
  __property int  id = {read=Getid, write=Setid};
  __property DOMString name = {read=Getname, write=Setname};
  __property DOMString phone = {read=Getphone, write=Setphone};
};
```

Ogni nodo figlio viene fatto corrispondere a una proprietà, il cui nome corrisponde al nome del marcatore del nodo figlio e il cui valore è l'interfaccia del nodo figlio (nel

caso il nodo figlio sia un nodo interno) o con il valore del nodo figlio (nel caso di nodi foglia). Anche ogni attributo di nodo viene fatto corrispondere con una proprietà, in cui il nome della proprietà è il nome dell'attributo e il valore della proprietà è il valore dell'attributo.

Oltre alla creazione di interfacce (e delle classi di implementazione discendenti) per ogni elemento contrassegnato incluso nel documento XML, il wizard crea una funzione globale per ottenere l'interfaccia verso il nodo radice. Ad esempio, se lo XML di cui sopra provenisse da un documento al cui nodo radice era associato il marcatore <Customers>, Data Binding wizard creerebbe le seguente routine globali:

```
extern PACKAGE _di_ICustomers __fastcall GetCustomers(TXMLDocument *XMLDoc);
extern PACKAGE _di_ICustomers __fastcall GetCustomers(_di_IXMLDocument XMLDoc);
extern PACKAGE _di_ICustomers __fastcall LoadCustomers(const WideString FileName);
extern PACKAGE _di_ICustomers __fastcall NewCustomers(void);
```

La funzione Get... accetta il wrapper dell'interfaccia di un'istanza *TXMLDocument* (o di un puntatore a quell'istanza di *TXMLDocument*). La funzione Load... crea dinamicamente un'istanza di *TXMLDocument* e carica il file XML specificato come suo valore prima di restituire un puntatore all'interfaccia. La funzione New... crea una nuova istanza (vuota) di *TXMLDocument* e restituisce l'interfaccia al nodo radice.

L'utilizzo delle interfacce generate semplifica il codice, perché riflettono la struttura del documento XML in modo più diretto. Ad esempio, invece di scrivere del codice simile al seguente:

```
_di_IXMLNode CustIntf = XMLDocument1->DocumentElement;
CustName = CustIntf->ChildNodes->Nodes->GetNode(0)->ChildNodes->Nodes[WideString("name")]-
>Value;
```

Il codice sarebbe invece:

```
_di_ICustomers CustIntf = GetCustomers(XMLDocument1);
CustName = CustIntf->Nodes->GetNode(0)->Name;
```

Si noti che le interfacce generate da Data Binding wizard discendono tutte da *IXMLNode*. Ciò significa che è sempre possibile aggiungere e cancellare i nodi figlio secondo le stesse modalità di quando non si utilizza Data Binding wizard. (Vedere "Aggiunta e cancellazione di nodi figlio" a pagina 35-5). Inoltre, quando i nodi figlio rappresentano elementi ripetitivi (quando tutti i figli di un nodo sono dello stesso tipo), al nodo genitore vengono assegnati due metodi, *Add* e *Insert*, per l'aggiunta di ulteriori ripetizioni. Questi metodi sono più semplici rispetto all'utilizzo di *AddChild*, perché non è necessario specificare il tipo di nodo da creare.

Utilizzo di XML Data Binding wizard

Per utilizzare Data Binding wizard,

1. Scegliere File | New | Other e selezionare l'icona con l'etichetta XML Data Binding nella zona inferiore della pagina New.
2. Viene visualizzato XML Data Binding wizard.

3 Sulla prima pagina del wizard specificare il documento o lo schema XML per cui si desidera generare interfacce. Questo può essere un documento XML di esempio, un file Document Type Definition (.dtd), un file Reduced XML Data (.xdr) o un file di schema XML (.xsd.

4 Fare clic sul pulsante Options per specificare le strategie per l'assegnazione dei nomi che si vuole vengano utilizzate dal wizard durante la generazione delle interfacce e delle classi di implementazione e la mappatura predefinita dei tipi definiti nello schema in tipi di dati nativi.

5 Passare alla seconda pagina del wizard. Questa pagina permette di fornire informazioni dettagliate su ogni tipo di nodo nel documento o nello schema. Alla sinistra c'è una vista ad albero che visualizza che tutto i tipi di nodo presenti nel documento. Per nodi complessi (nodi che hanno figli), la vista albero può essere estesa in modo da visualizzare gli elementi figlio. Quando si seleziona un nodo nella vista ad albero, il lato destro della finestra di dialogo visualizza le informazioni su quel nodo e permette di specificare come si desidera che il wizard tratti quel nodo.

- Il controllo Source Name visualizza il nome del tipo di nodo nello schema XML.
- Il controllo Source Datatype visualizza il tipo del valore del nodo, come specificato nello schema XML.
- Il controllo Documentation permette di aggiungere commenti allo schema per descrivere l'utilizzo o lo scopo del nodo.
- Se il wizard genera del codice per il nodo selezionato (cioè, se è un tipo complesso per il quale il wizard genera un'interfaccia e una classe di implementazione oppure se è uno degli elementi figlio di un tipo complesso per il quale il wizard genera una proprietà sull'interfaccia del tipo complesso), è possibile utilizzare la casella di controllo Generate Binding per specificare se si desidera che il wizard generi codice per il nodo. Se si deseleziona Generate Binding, il wizard non genera l'interfaccia o la classe di implementazione per un tipo complesso oppure non crea una proprietà nell'interfaccia genitore di un elemento o di un attributo figlio.
- La sezione Binding Options permette di influire sul codice generato dal wizard per l'elemento selezionato. Per qualsiasi nodo, è possibile specificare l'Identifier Name (il nome dell'interfaccia o della proprietà generata).Inoltre, per le interfacce, è necessario indicare quale rappresenta il nodo radice del documento. Per nodi che rappresentano proprietà, è possibile specificare il tipo della proprietà e, se la proprietà non è un'interfaccia, se è una proprietà a sola lettura.

6 Una volta specificato quale codice si desidera che il wizard generi per ogni nodo, passare alla terza pagina. Questa pagina permette di scegliere alcune opzioni globali relative a come il wizard genera il codice, permette di vedere in anteprima il codice che sarà generato e permette di dire al wizard come salvare le scelte fatte per un successivo utilizzo.

- Per vedere in anteprima il codice generato dal wizard, selezionare un'interfaccia nella lista Binding Summary e visualizzare la definizione di interfaccia risultante nel controllo Code Preview.

- Utilizzare Data Binding Settings per indicare come il wizard dovrebbe salvare le scelte. È possibile memorizzare le impostazioni come annotazioni in un file di schema, associato al documento (il file di schema specificato nella prima pagina della finestra di dialogo) oppure è possibile dare un nome a un file di schema indipendente, utilizzato solo dal wizard.

7 Appena si fa clic su Finish, Data Binding wizard genera una nuova unit che definisce le interfacce e le classi di implementazione per tutti i tipi di nodo presenti nel documento XML. Inoltre, crea una funzione globale che prende un oggetto *TXMLDocument* e restituisce l'interfaccia per il nodo radice della gerarchia dati.

Utilizzo del codice generato da XML Data Binding wizard

Una volta che il wizard ha generato un insieme di interfacce e di classi di implementazione, è possibile utilizzarle per operare con documenti XML che corrispondono alla struttura del documento o dello schema fornito al wizard. Esattamente come quando si utilizzano solo i componenti XML nativi, il punto di partenza è il componente *TXMLDocument* presente sulla pagina Internet della Component palette.

Per operare con un documento XML, fare quanto segue:

1 Ottenere un'interfaccia per il nodo radice del documento XML. È possibile farlo in tre modi:

- Collocare un componente *TXMLDocument* sulla scheda o nel modulo dati. Collegare il componente *TXMLDocument* a un documento XML impostando la proprietà *FileName*. (Come metodo alternativo, è possibile utilizzare una stringa XML impostando la proprietà *XML* in esecuzione). Poi, nel codice, chiamare la funzione globale creata dal wizard per ottenere un'interfaccia per il nodo radice del documento XML. Ad esempio, se l'elemento radice del documento XML era il marcatore <StockList>, per impostazione predefinita, il wizard genera una funzione *GetStockListType*, che restituisce un'interfaccia *IStockListType*:

```
XMLDocument1->FileName := "Stocks.xml";
_di_IStockListType StockList = GetStockListType(XMLDocument1);
```

- Chiamare la funzione generata Load... per creare e collegare l'istanza di *TXMLDocument* e ottenere la sua interfaccia in un solo passaggio. Ad esempio, utilizzando lo stesso documento XML descritto in precedenza:

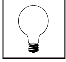
```
_di_IStockListType StockList = LoadStockListType("Stocks.xml");
```

- Chiamare la funzione New... generata per creare l'istanza di *TXMLDocument* di un documento vuoto quando si vogliono creare tutti i dati nell'applicazione:

```
_di_IStockListType StockList = NewStockListType();
```

2 Questa interfaccia ha proprietà che corrispondono ai sottonodi dell'elemento radice del documento, oltre a proprietà che corrispondono agli attributi di quell'elemento radice. È possibile utilizzare queste proprietà per percorrere la gerarchia del documento XML, modificare i dati nel documento, e così via.

3 Per salvare tutte le modifiche apportate utilizzando le interfacce generate dal wizard, chiamare il metodo *SaveToFile* del componente *TXMLDocument* o leggerne la proprietà *XML*.



Se si imposta la proprietà *Options* dell'oggetto *TXMLDocument* in modo da includere *doAutoSave* non è necessario chiamare esplicitamente il metodo *SaveToFile*.

Capitolo

36

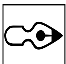
Utilizzo dei Web Services

I Web Services sono applicazioni modulari indipendenti che possono essere pubblicate e richiamate da Internet. I Web Services mettono a disposizione interfacce ben definite che descrivono i servizi forniti. A differenza delle applicazioni per Web server che generano pagine web per i browser client, le applicazioni Web Services non sono progettate per l'interazione diretta con l'utente. Ad esse invece si accede mediante il codice dalle applicazioni client.

I Web Services sono progettati per consentire un accoppiamento flessibile tra client e server. Ciò significa che le implementazioni server non chiedono che i client utilizzino una piattaforma o un linguaggio di programmazione specifici. Oltre a definire le interfacce in un modo indipendente dal linguaggio, essi sono progettati per consentire anche più meccanismi di comunicazione.

Il supporto di C++Builder per i Web Services è stato concepito per funzionare utilizzando SOAP (Simple Object Access Protocol). SOAP è un protocollo standard leggero per lo scambio di informazioni in un ambiente decentrato e distribuito. Esso utilizza XML per codificare le chiamate a procedura remote e, come protocollo di comunicazione, di solito utilizza HTTP. Per ulteriori informazioni su SOAP, vedere le specifiche SOAP disponibili all'indirizzo

```
http://www.w3.org/TR/SOAP/
```



Benché i componenti che supportano i Web Services siano stati concepiti per utilizzare SOAP e HTTP, il framework è sufficientemente generico da poter essere ampliato in modo da utilizzare altri protocolli di codifica e di comunicazione.

Oltre a consentire la costruzione di applicazioni Web Service basate su SOAP (server), C++Builder fornisce il supporto per i client dei Web Services che utilizzano una codifica SOAP o uno stile Document Literal. Lo stile Document Literal è utilizzato nei Web Services .Net.

I componenti che supportano i Web Services sono disponibili sia in Windows che in Linux, e pertanto è possibile utilizzarli come elementi base per applicazioni distribuite multipiattaforma. Non occorre installare alcun particolare software client runtime, come quando si distribuiscono le applicazioni che utilizzano CORBA.

Poiché questa tecnologia si basa sui messaggi HTTP, presenta il vantaggio che è ampiamente disponibile su una vasta gamma di macchine. Il supporto per i Web Services si fonda sull'architettura di applicazioni per Web server (Web Broker).

Le applicazioni Web Service pubblicano mediante un documento WSDL (Web Service Definition Language) informazioni sulle interfacce disponibili e su come chiamarle. Sul lato server, l'applicazione può pubblicare un documento WSDL che descrive il Web Service. Sul lato client, un wizard o programma di utilità a riga di comando può importare un documento WSDL pubblicato, fornendo le definizioni delle interfacce e le informazioni di connessione necessarie. Se già si possiede un documento WSDL che descrive il servizio Web che si desidera implementare, è possibile generare il codice lato server anche quando si importa il documento WSDL.

Le interfacce richiamabili

I server che supportano i Web Services sono costruiti utilizzando interfacce richiamabili. Le interfacce richiamabili sono classi pure virtual (classi che includono solo metodi pure virtual) che vengono compilate in modo da includere informazioni di tipo in esecuzione (RTTI). Sul server, queste RTTI vengono utilizzate durante l'interpretazione delle chiamate ai metodo in arrivo dai client in modo che possano essere smistate correttamente. Sui client, queste RTTI vengono utilizzate per generare dinamicamente una tabella dei metodi per effettuare chiamate ai metodi dell'interfaccia.

Per creare un'interfaccia richiamabile, è necessario dichiarare la classe dell'interfaccia utilizzando la parola chiave **_declspec** , utilizzando il modificatore *delphirtti*. Anche il discendente di qualsiasi interfaccia richiamabile è a sua volta richiamabile. Tuttavia, se un'interfaccia richiamabile discende da un'altra classe di interfaccia che non è richiamabile, il Web Service può solo utilizzare i metodi definiti nell'interfaccia richiamabile e nei suoi discendenti. I metodi ereditati da antenati non richiamabili non vengono compilati con le informazioni di tipo e non possono essere così utilizzati come parte del Web Service.



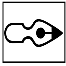
Per informazioni sulle classi di interfaccia in C++Builder, vedere "Eredità e interfacce" a pagina 13-2.

Il file header sysmac.h definisce un'interfaccia richiamabile di base, *IInvokable*, da cui è possibile derivare qualsiasi interfaccia esposta ai client da un server Web Service. *IInvokable* è identica all'interfaccia di base (*IInterface*), tranne per il fatto che è compilata utilizzando l'opzione **_declspec** (delphirtti*)* in modo che essa e tutti i suoi discendenti includano le RTTI.

Ad esempio, il seguente codice definisce un'interfaccia richiamabile che contiene due metodi per codificare e decodificare valori numerici:

```
__interface INTERFACE_UUID("{C527B88F-3F8E-1134-80e0-01A04F57B270}") IEncodeDecode :
    public IInvokable
{
public:
  virtual double __stdcall  EncodeValue(int Value) = 0 ;
  virtual int __stdcall  DecodeValue(double Value) = 0 ;
```

```
};
```



In questo esempio, si noti l'utilizzo di **__interface** nella dichiarazione. Questa non è una vera parola chiave ma piuttosto una macro che viene utilizzata per convenzione con le interfacce. È mappata sulla parola chiave **class**. La macro INTERFACE_UUID assegna all'interfaccia un identificativo globalmente univoco (GUID). Si noti che la classe dell'interfaccia contiene solo metodi pure virtual.

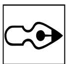
Prima che un'applicazione Web Service possa utilizzare questa interfaccia richiamabile, deve essere registrata con il file di registro delle chiamate. Sul server, l' elemento del registro delle chiamate consente al componente invoker (*THTTPSOAPCppInvoker*) di identificare una classe di implementazione da utilizzare per l'esecuzione delle chiamate all'interfaccia. Sulle applicazioni client, un elemento del registro delle chiamate permette agli oggetti remoti interfacciati (*THTTPRio*) di cercare informazioni che identifichino l'interfaccia richiamabile e forniscano informazioni su come chiamarla.

Di solito, il client o il server Web Service crea il codice per definire le interfacce richiamabili o importando un documento WSDL o utilizzando il Web Service wizard. Per impostazione predefinita, quando l'importatore di WSDL o il Web Service wizard generano un'interfaccia, la definizione viene aggiunta a un file header avente lo stesso nome del Web Service. Il corrispondente file .cpp contiene il codice per registrare l'interfaccia (oltre che una classe di implementazione nel caso si stia scrivendo un server). Per l'interfaccia descritta in precedenza, questo codice di registrazione assomiglia al seguente:

```
static void RegTypes()
{
  InvRegistry()->RegisterInterface(__delphirtti(IEncodeDecode), "", "");
}

#pragma startup RegTypes 32
```

Le interfacce dei Web Service devono avere un namespace per identificarli tra tutte le interfacce in tutti i possibili Web Service. L'esempio precedente non fornisce un namespace per l'interfaccia. Quando non si fornisce esplicitamente un namespace, il file di registro delle chiamate ne genera automaticamente uno. Questo namespace è costituito da una stringa che identifica l'applicazione in modo univoco (la variabile AppNamespacePrefix), dal nome dell'interfaccia e dal nome della unit in cui è definita. Se non si desidera utilizzare il namespace generato in automatico, è possibile specificarne uno utilizzando esplicitamente un secondo parametro nella chiamata a RegisterInterface.



Non tentare di utilizzare lo stesso file header che definisce un'interfaccia richiamabile sia nelle applicazioni client che server. Questo può provocare facilmente incongruenze nei namespace, perché quando il file header viene incluso in un altro file sorgente, il nome della unit viene modificato in quello del file sorgente che include, modificando così il nome del namespace generato. Pertanto, le applicazioni client dovrebbero importare il Web Service utilizzando un documento WSDL.

Utilizzo di tipi non scalari nelle interfacce richiamabili

L'architettura dei Web Services include automaticamente il supporto per lo smistamento dei tipi scalari seguenti:

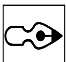
- bool
- char
- signed char
- unsigned char
- short
- unsigned short
- int
- unsigned int
- long
- unsigned long
- __int64
- unsigned __int64
- float
- double
- long double
- AnsiString
- WideString
- Currency
- Variant

Non sono necessarie operazioni particolari quando si utilizzano questi tipi in un'interfaccia richiamabile. Tuttavia, se l'interfaccia include qualche proprietà o metodo che utilizza altri tipi, l'applicazione deve registrare quei tipi con il registro dei tipi remotable. Per ulteriori informazioni sul registro dei tipi remotable, vedere "Registrazione di tipi non scalari" a pagina 36-5.

I tipi enumerati e i tipi dichiarati con un'istruzione typedef richiedono una piccola operazione supplementare in modo che il registro dei tipi remotable possa estrarre dalla definizione del tipo le informazioni di tipo di cui ha bisogno. Queste sono descritte nella sezione "Registrazione di tipi dichiarati con typedef e di tipi enumerati" a pagina 36-6.

I tipi dynamic array definiti in sysdyn.h vengono registrati automaticamente, e pertanto l'applicazione non deve aggiungere alcun codice di registrazione particolare. Uno di questi in particolare, *TByteDynArray*, merita una menzione speciale poiché mappa un blocco di dati binari 'base64', invece di mappare separatamente ogni elemento dell'array come fanno gli altri tipi di dynamic array.

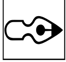
Per tutti gli altro tipi, come static array, struct o classi, è necessario mappare il tipo con una classe remotable. Una classe remotable è una classe che include informazioni di tipo in esecuzione (RTTI). L'interfaccia deve utilizzare quindi la classe remotable invece dell'array statico corrispondente, dello struct o della classe. Qualsiasi classe remotable si crei, la si deve registrare con il registro dei tipi remotable.



Tutti i tipi dovrebbero essere dichiarati esplicitamente con un'istruzione **typedef** invece che essere dichiarati in linea. Ciò è necessario in modo che il registro dei tipi remotable possa determinare il nome del tipo C++ nativo.

Registrazione di tipi non scalari

Prima che un'interfaccia richiamabile possa utilizzare un qualsiasi tipo diverso dai tipi scalari mativi elencati nella sezione "Utilizzo di tipi non scalari nelle interfacce richiamabili" a pagina 36-4, l'applicazione deve registrare il tipo con il registro dei tipi remotable. Per accedere al registro dei tipi remotable, è necessario includere InvokeRegistry.hpp nel file sorgente. Questo header dichiara una funzione globale, *RemTypeRegistry* (), che restituisce un riferimento al registro dei tipi remotable.



Sui client, il codice per registrare i tipi con il registro di tipi remotable viene generato automaticamente quando si importa un documento WSDL. Per i server, i tipi remotable vengono registrati automaticamente quando si registra un'interfaccia che li utilizza. È necessario aggiungere esplicitamente il codice per registrare tipi solo se si desidera specificare il namespace o il nome del tipo invece di utilizzare i valori generati in automatico.

Il registro dei tipi remotable ha due metodi che è possibile utilizzare per registrare i tipi: *RegisterXSInfo* e *RegisterXSClass*. Il primo (*RegisterXSInfo*) permette di registrare un array dinamico. Il secondo (*RegisterXSClass*) serve per la registrazione di classi remotable che vengono definite per rappresentare altri tipi.

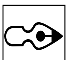
Se si utilizzano array dinamici, il registro delle chiamate può ottenere le informazioni di cui ha bisogno dalle informazioni di tipo generate dal compilatore. Così, ad esempio, l'interfaccia può utilizzare un tipo come quello seguente:

```
typedef DynamicArray<TXSDateTime> TDateTimeArray;
```

Questo tipo viene registrato automaticamente quando si registra l'interfaccia richiamabile. Tuttavia, se si desidera specificare il namespace in cui il tipo è definito o il nome del tipo, è necessario aggiungere la seguente registrazione alla funzione RegTypes usata per registrare l'interfaccia richiamabile che utilizza questo array dinamico:

```
void RegTypes()
{

  RemTypeRegistry()->RegisterXSInfo(__arraytypeinfo(TDateTimeArray),
        MyNameSpace, "DTarray", "DTarray");
  InvRegistry()->RegisterInterface(__delphirtti(ITimeServices));
}
```



Non utilizzare più tipi dynamic array che mappano tutti lo stesso tipo di elemento sottostante. Poiché il compilatore li tratta come tipi trasparenti che possono essere convertiti implicitamente in un altro tipo, non è in grado di distinguere le rispettive informazioni di tipo in esecuzione.

Il primo parametro di *RegisterXSInfo* è l'informazione di tipo per il tipo che si sta registrando. Il secondo parametro è l'URI del namespace per il namespace in cui il tipo è definito. Se si omette questo parametro o si fornisce una stringa vuota, il registro genera automaticamente un namespace. Il terzo parametro è il nome del tipo come appare nel codice nativo C++. È necessario fornire un valore per questo parametro. L'ultimo parametro è il nome del tipo come appare nei documenti WSDL. Se si omette questo parametro o si fornisce una stringa vuota, il registro utilizza il nome del tipo nativo (il terzo parametro).

La registrazione di una classe remotable è simile, tranne per il fatto che si fornisce un riferimento di classe invece che un puntatore alle informazioni di tipo. Ad esempio, il seguente codice registra una classe remotable di nome *TXSRecord*:

```
void RegTypes()
{
  RemTypeRegistry()->RegisterXSclass(__classid(TXSRecord), MyNameSpace, "record", "",
false);
  InvRegistry()->RegisterInterface(__delphirtti(ITimeServices));
}
```

Il primo parametro è il riferimento di classe per la classe remotable che rappresenta il tipo. Il secondo è un identificativo uniforme di risorse (URI) che identifica in modo univoco il namespace della nuova classe. Se si fornisce una stringa vuota, il registro genera automaticamente un URI. Il terzo e il quarto parametro specificano i nomi nativi ed esterni del tipo di dati che la classe rappresenta. Se si omette il secondo e il quarto parametro, il registro dei tipi utilizza il terzo parametro per entrambi i valori. Se si fornisce una stringa vuota per entrambi i parametri, il registro utilizza il nome della classe. Il quinto parametro indica se il valore delle istanze della classe può essere trasmesso come stringa. Volendo è possibile aggiungere un sesto parametro (non visualizzato qui) per controllare in che modo dovrebbero essere rappresentati nei pacchetti SOAP più riferimenti alla stessa istanza dell'oggetto.

Registrazione di tipi dichiarati con typedef e di tipi enumerati

Se l'interfaccia richiamabile utilizza un tipo enumerato, la funzione **__delphirtti** non è in grado di estrarre direttamente le sue informazioni di tipo. Per aggirare l'ostacolo, è necessario generare una classe contenitore in stile VCL che può essere utilizzata per estrarre le informazioni di tipo per il tipo enumerato:

```
class MyEnumType_TypeInfoHolder : public TObject {
  MyEnumType __instanceType;
public:
__published:
  __property MyEnumType __propType = {read=__instanceType };
};
```

In questo caso, la classe *MyEnumType_TypeInfoHolder* viene definita in modo da estrarre le informazioni di tipo da un tipo enumerato di nome *MyEnumType*. Ha una singola proprietà published, *__propType*, il cui tipo è il tipo enumerato che si desidera registrare. Una volta definita la classe contenitore, è possibile ottenere le informazioni di tipo per il tipo enumerato chiamando la funzione globale *GetClsMemberTypeInfo*. Il seguente codice illustra come registrare un tipo enumerato, data la sua classe detentrice:

```
void RegTypes()
{
  RemTypeRegistry()->RegisterXSInfo(
          GetClsMemberTypeInfo(__classid(MyEnumType_TypeInfoHolder), "__propType"),
          MyNameSpace, "MyEnumType");
}
```

RegisterClsMemberTypeInfo ha due parametri: Le informazioni di tipo della classe contenitrice e il nome della proprietà pubblicata dal cui tipo si desidera estrarre le

informazioni di tipo. Se la classe contenitrice ha solo una proprietà pubblicata (come nell'esempio precedente), è possibile omettere il secondo parametro.

Questa tecnica di utilizzare una classe contenitrice deve essere utilizzata anche se il tipo è dichiarato utilizzando un'istruzione typedef che lo mappa con un tipo scalare nativo di C++. (un tipo scalare nativo di C++ è un qualunque tipo fra quelli elencati in "Utilizzo di tipi non scalari nelle interfacce richiamabili" a pagina 36-4 tranne per quelli che sono classi che emulano i tipi Object Pascal). Perciò, ad esempio, dato il seguente tipo:

```
typedef int CardNumber;
```

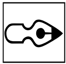
Si dovrebbe creare una classe contenitrice come quella seguente:

```
class CardNumberType_TypeInfoHolder : public TObject {
  CardNumber __instanceType;
public:
__published:
  __property CardNumber __propType = {read=__instanceType };
};
```

e quindi registrarla come segue:

```
RemTypeRegistry()->RegisterXSInfo(GetClsMemberTypeInfo(
          __classid(CardNumber_TypeInfoHolder), "__propType"),
          MyNameSpace, "CardNumber");
```

Per qualsiasi altro tipo dichiarato utilizzando un'istruzione typedef, chiamare *RegisterXSInfo* se è mappato su una classe di array dinamico, e *RegisterXSClass* se è mappato con una classe. Per ulteriori informazioni sulla registrazione di array dinamici e classi, vedere "Registrazione di tipi non scalari" a pagina 36-5.



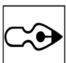
Si dovrebbe cercare di evitare istruzioni typedef che mappano più tipi con lo stesso tipo sottostante. Le informazioni di tipo in esecuzione per questi non sono distinguibili.

Utilizzo di oggetti remotable

Utilizzare *TRemotable* come classe di base quando si definisce una classe per rappresentare un tipo di dati complesso su un'interfaccia richiamabile. Ad esempio, nel caso in cui di solito si passerebbe come parametro uno struct, si potrebbe invece definire un discendente *Tremotable*, in cui ogni membro dello struct è una proprietà pubblicata della nuova classe.

È possibile controllare se le proprietà pubblicate del discendente *TRemotable* appaiono come nodi di elemento o attributi nella corrispondente codifica SOAP del tipo. Per trasformare una proprietà in un attributo, utilizzare la direttiva **stored** sulla definizione della proprietà, assegnando un valore di AS_ATTRIBUTE:

```
__property bool MyAttribute =
                         {read=FMyAttribute, write=FMyAttribute, stored= AS_ATTRIBUTE;
```



Se non si include una direttiva **stored** o se si assegna un qualsiasi altro valore alla direttiva stored (anche una funzione che restituisce AS_ATTRIBUTE), la proprietà viene codificata come un nodo piuttosto che come un attributo.

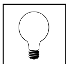
Se il valore del nuovo discendente *TRemotable* rappresenta un tipo scalare in un documento WSDL, come classe di base si dovrebbe utilizzare invece *TRemotableXS*. *TRemotableXS* è un discendente di *TRemotable* che introduce due metodi per la conversione della nuova classe nella sua rappresentazione di stringa. Implementare questi metodi ridefinendo i metodi *XSToNative* e *NativeToXS*.

Per alcuni tipi scalari di XML comunemente usati, la unit XSBuiltIns definisce e registra automaticamente una serie di classi remotable. Queste sono elencate nella seguente tabella:

Tabella 36.1 Classi Remotable

Tipo XML	classe remotable
dateTime timeInstant	TXSDateTime
data	TXSDate
tempo	TXSTime
durata timeDuration	TXSDuration
decimale	TXSDecimal
hexBinary	TXSHexBinary

Dopo avere definito una classe remotable, la si deve registrare con il registro dei tipi remotable, come descritto in "Registrazione di tipi non scalari" a pagina 36-5.Questa registrazione avviene automaticamente su server quando si registra l'interfaccia che utilizza la classe. Sui client, il codice per registrare la classe viene generato automaticamente quando si importa il documento WSDL che definisce il tipo.



È buona norma implementare e registrare i discendenti *TRemotable* in una unit separata dal resto dell'applicazione server, includendo dalle unit che dichiarano e registrano le interfacce richiamabili. In questo modo, è possibile utilizzare il tipo per più di un'interfaccia.

Un problema che si presenta quando si utilizzano discendenti di *TRemotable* è il momento in cui tali discendenti vengono creati e distrutti. Ovviamente, l'applicazione server deve creare la propria istanza locale di questi oggetti, poiché l'istanza del chiamante è in uno spazio di processo separato. Per gestire ciò, le applicazioni Web Services creano un contesto di dati per le richieste in arrivo. Il contesto di dati persiste mentre il server gestisce la richiesta e viene liberato dopo che ogni parametro di output è stato smistato in un messaggio di ritorno. Quando il server crea istanze locali di oggetti remotable, li aggiunge al contesto di dati e quelle istanze vengono quindi liberate insieme al contesto di dati. In alcuni casi, si potrebbe volere evitare che un'istanza di un oggetto remotable venga liberata dopo una chiamata a metodo. Ad esempio, se l'oggetto contiene informazioni di stato, potrebbe essere più efficiente avere una singola istanza che viene utilizzato per ogni chiamata di messaggio. Per evitare che l'oggetto remotable venga liberato insieme al contesto di dati, modificarne la proprietà *DataContext*.

Esempio di oggetto Remotable

Questo esempio mostra come creare un oggetto remotable per un parametro su un'interfaccia richiamabile dove altrimenti si utilizzerebbe una classe esistente. In questo esempio, la classe esistente è un elenco di stringhe (*TStringList*). Per limitare le dimensioni dell'esempio, non viene riprodotta la proprietà *Objects* dell'elenco di stringhe.

Poiché la nuova classe non è scalare, essa deriva da *TRemotable* invece che da *TRemotableXS*. Include una proprietà pubblicata per ogni proprietà dell'elenco di stringhe che si desidera comunicare tra il client e il server. Ognuna di queste proprietà remotable corrisponde a un tipo remotable. Inoltre, la nuova classe remotable include metodi per effettuare la conversione da un elenco di stringhe e viceversa.

```
class TRemotableStringList: public TRemotable
{
  private:
    bool FCaseSensitive;
    bool FSorted;
    Classes::TDuplicates FDuplicates;
    System::TStringDynArray FStrings;
  public:
    void __fastcall Assign(Classes::TStringList *SourceList);
    void __fastcall AssignTo(Classes::TStringList *DestList);
__published:
    __property bool CaseSensitive = {read=FCaseSensitive, write=FCaseSensitive};
    __property bool Sorted = {read=FSorted, write=FSorted};
    __property Classes::TDuplicates Duplicates = {read=FDuplicates, write=FDuplicates};
    __property System::TStringDynArray Strings = {read=FStrings, write=FStrings};
}
```

Si noti che *TRemotableStringList* esiste solo come classe di trasporto. Pertanto, benché abbia una proprietà *Sorted* (per trasportare il valore di una proprietà *Sorted* di un elenco di stringhe), non è necessario che ordini le stringhe che memorizza ma deve solo memorizzare se le stringhe devono essere ordinate o meno. In questo modo l'implementazione risulta molto semplice. È necessario solo implementare i metodi *Assign* e *AssignTo* , che si convertono da un elenco di stringhe e viceversa:

```
void __fastcall TRemotableStringList::Assign(Classes::TStringList *SourceList)
{
  SetLength(Strings, SourceList->Count);
  for (int i = 0; i < SourceList->Count; i++)
    Strings[i] = SourceList->Strings[i];
  CaseSensitive = SourceList->CaseSensitive;
  Sorted = SourceList->Sorted;
  Duplicates = SourceList->Duplicates;
}

void __fastcall TRemotableStringList::AssignTo(Classes::TStringList *DestList)
{
  DestList->Clear();
  DestList->Capacity = Length(Strings);
  DestList->CaseSensitive = CaseSensitive;
  DestList->Sorted = Sorted;
```

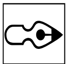
```
    DestList->Duplicates = Duplicates;
    for (int i = 0; i < Length(Strings); i++)
      DestList->Add(Strings[i]);
  }
```

Facoltativamente, si potrebbe volere registrare la nuova classe remotable in modo che sia possibile specificare il suo nome di classe. Se non si registra la classe, essa viene registrato automaticamente quando si registra l'interfaccia che la utilizza. Allo stesso modo, se si registrano la classe ma non i tipi *TDuplicates* e *TStringDynArray* che essa utilizza, essi vengono registrati automaticamente. Il codice seguenete mostra come registrare la classe *TRemotableStringList*. *TStringDynArray* viene registrato automaticamente perché è uno dei tipi array dinamici nativi dichiarati in sysdyn.h. Per i dettagli sulla registrazione esplicita di un tipo enumerato come *TDuplicates*, vedere "Registrazione di tipi dichiarati con typedef e di tipi enumerati" a pagina 36-6.

```
void RegTypes()
{
  RemTypeRegistry()->RegisterXSclass(__classid(TRemotableStringList), MyNameSpace,
"stringList", "", false);
}
#pragma startup initServices 32
```

Scrittura di server che supportano i Web Services

Oltre alle interfacce richiamabili e alle classi discendenti che implementano i rispettivi metodi pure virtual, il server richiede due componenti: un dispatcher e un invoker. Il dispatcher (*THTTPSoapDispatcher*) riceve i messaggi SOAP in arrivo e li passa all'invoker. L'invoker (*THTTPSoapCppInvoker*) interpreta il messaggio SOAP, identifica l'interfaccia richiamabile che esso chiama, esegue la chiamata e assembla il messaggio di risposta.



THTTPSoapDispatcher e *THTTPSoapCppInvoker* sono progettati per rispondere a messaggi HTTP che contengono una richiesta SOAP. L'architettura sottostante è sufficientemente generale, tuttavia, che è in grado di supportare altri protocolli che utilizzano componenti dispatcher e invoker differenti.

Una volta che si registrano le interfacce richiamabili e le loro classi di implementazione, il dispatcher e l'invoker gestiscono automaticamente qualsiasi messaggio che identifica quelle interfacce nell'Action header di SOAP Action del messaggio di richiesta HTTP.

I Web Service includono anche un publisher (*TWSDLHTMLPublish*). I publisher rispondono alle richieste dei client in arrivo creando i documenti WSDL che descrivono come chiamare i Web Services nell'applicazione.

Creazione di un server Web Service

Per creare un'applicazione server che implementa un Web Service fare quanto segue:

1 Scegliere File | New | Other e sulla pagina WebServices, fare doppio clic sull'icona Soap Server Application per avviare il SOAP Server Application wizard. Il wizard crea una nuova applicazione Web server che include i componenti necessari per rispondere alle richieste SOAP. Per i dettagli sul wizard per le applicazioni SOAP e sul codice che esso genera, vedere "Utilizzo di SOAP application wizard" a pagina 36-11.

2 Quando si esce da SOAP Server Application wizard, viene chiesto se si desidera definire un'interfaccia per il Web Service. Se si sta creano un web Service da zero, facendo clic su Yes si vedrà apparire il wizard Add New Web Service. Il wizard aggiunge il codice per dichiarare e registrare una nuova interfaccia richiamabile per il Web Service. Editare il codice generato per definire e implementare il Web Service. Se si desidera aggiungere altre interfacce (o si desidera definire le interfacce in un momento successivo), scegliere File | New | Other e, sulla pagina WebServices, fare doppio clic sull'icona SOAP Web Service interface. Per i dettagli sull'uso del wizard Add New Web Service e sul completamento del codice generato, vedere "Aggiunta di nuovi Web Services" a pagina 36-12.

3 Se si sta implementando un Web Service che è stato già definito in un documento WSDL, è possibile utilizzare Web Services Importer per generare le interfacce, le classi di implementazione e il codice di registrazione necessari all'applicazione. È sufficiente solo completare il corpo dei metodi che l'importatore genera per le classi di implementazione. Per i dettagli sull'utilizzo di Web Services Importer, vedere "Utilizzo di Web Services Importer" a pagina 36-14.

4 Se l'applicazione solleva un'eccezione nel momento in cui tenta di eseguire una richiesta SOAP, l'eccezione verrà codificata automaticamente in un pacchetto di errore SOAP, che sarà restituito al posto dei risultati della chiamata al metodo. Se si vogliono ottenere maggiori informazioni al posto di un semplice messaggio di errore, è posisbile creare delle classi di eccezione che saranno codificate e passate al client. L'operazione è descritta in "Creazione di classi di eccezione custom per Web Services" a pagina 36-15.

5 SOAP Server Application wizard aggiunge alle nuove applicazioni Web Service un componente publisher (*TWSDLHTMLPublish*). Ciò permette all'applicazione di pubblicare documenti WSDL che descrivono il Web Service ai client. Per informazioni sul publisher WSDL, vedere "Generazione di documenti WSDL per un'applicazione Web Service" a pagina 36-16.

Utilizzo di SOAP application wizard

Le applicazioni Web Service sono una forma particolare di applicazioni Web Server. Per questo motivo, il supporto per Web Services è costruito sopra all'architettura Web Broker. Pertanto, per capire il codice generato da SOAP Application wizard, è utile per capire l'architettura Web Broker. Le informazioni sulle applicazioni per Web server in generale e su Web Broker in particolare, sono reperibili nel Capitolo 32, "Creazione di applicazioni server per Internet" e nel Capitolo 33, "Utilizzo di Web Broker".

Per avviare SOAP application wizard, scegliere File | New | Other e sulla pagina WebServices, fare doppio clic sull'icona Soap Server Application. Scegliere il tipo di

applicazione per Web server che si desidera utilizzare per il Web Service. Per informazioni sui vari tipi di applicazioni per Web server, vedere “Tipi di applicazioni per server Web” a pagina 32-7.

Il wizard genera una nuova applicazione per Web server che include un modulo Web che contiene tre componenti:

- Un componente invoker (*THTTPSOAPCppInvoker*). L’invoker effettua la conversione tra i messaggi SOAP e i metodi di una qualunque interfaccia richiamabile registrata nell’applicazione Web Service.
- Un componente dispatcher (*THTTPSoapDispatcher*). Il dispatcher risponde automaticamente ai messaggi SOAP in arrivo e li invia all’invoker. È possibile utilizzare la sua proprietà *WebDispatch* per identificare i messaggi di richiesta HTTP a cui l’applicazione risponde. Ciò implica l’impostazione della proprietà *PathInfo* per indicare la parte di percorso di un qualunque URL indirizzato all’applicazione e della proprietà *MethodType* per indicare l’header del metodo per i messaggi di richiesta.
- Un editor WSDL (*TWSDLHTMLPublish*). L’editor WSDL pubblica un documento WSDL che descrive le interfacce e la modalità con cui chiamarle. Il documento WSDL dice ai client come effettuare la chiamata all’applicazione Web Service. Per i dettagli sull’utilizzo dell’editor WSDL, vedere “Generazione di documenti WSDL per un’applicazione Web Service” a pagina 36-16.

Il dispatcher SOAP e l’editor WSDL sono componenti auto-dispatching. Questo significa che essi si registrano automaticamente con il modulo Web in modo che inoltri qualsiasi richiesta in arrivo indirizzata utilizzando le informazioni di percorso che essi specificano nelle rispettive proprietà *WebDispatch*. Se si fa clic destro sul modulo Web, è possibile vedere che, oltre a questi componenti auto-dispatching, ha un singolo elemento di azione Web di nome *DefaultHandler*.

DefaultHandler è l’elemento di azione predefinito. Cioè, se il modulo Web riceve una richiesta per la quale non è in grado di trovare un gestore (non è in grado di trovare una corrispondenza con le informazioni di percorso), invia quel messaggio all’elemento di azione predefinito. *DefaultHandler* genera una pagina web che descrive il Web Service. Per modificare l’azione predefinita, modificare il gestore di evento *OnAction* dell’elemento di azione.

Aggiunta di nuovi Web Services

Per aggiungere all’applicazione server una nuova interfaccia Web Service, scegliere File | New | Other e sulla pagina WebServices fare doppio clic sull’icona SOAP Server Interface.

Il wizard Add New Web Service consente di specificare il nome dell’interfaccia richiamabile che si desidera esporre ai client e genera il codice per dichiarare e registrare l’interfaccia e la sua classe di implementazione da essa derivata. Per impostazione predefinita, il wizard genera anche commenti che mostrano i metodi di esempio e le definizioni di tipo aggiuntive, che aiutano a iniziare l’editing dei file generati.

Editing del codice generato

Le definizioni dell'interfaccia appaiono nel file header della unit generata, il cui nome è quello che era stato specificato utilizzando il wizard. Si vorrà modificare la dichiarazione dell'interfaccia, sostituendo i metodi di esempio con i metodi che si stanno rendendo disponibili ai client.

Il wizard genera una classe di implementazione che discende da *TInvokableClass* e dall'interfaccia richiamabile. Se si sta definendo da zero un'interfaccia richiamabile, è necessario modifocare la dichiarazione della classe di implementazione in kodo che corrisponda alle eventuali modifiche apportate all'interfaccia richiamabile generata.

Quando si aggiungono metodi all' interfaccia richiamabile e alla classe di implementazione, è bene ricordare che i metodi devono utilizzare solo tipi remotable. Per informazioni sui tipi remotable e sulle interfacce richiamabili, vedere "Utilizzo di tipi non scalari nelle interfacce richiamabili" a pagina 36-4.

Utilizzo di una classe di base differente

Il wizard Add New Web Service genera classi di implementazione che discendono da *TInvokableClass*. Questo è il modo più semplice per creare una nuova classe per implementare un Web Service. È possibile, tuttavia, sostituire questa classe generata con una classe di implementazione che ha una classe di base differente (ad esempio, si potrebbe volere utilizzare come una classe base una classe esistente). Quando si sostituisce la classe di implementazione generata, è bene fare alcune considerazioni:

- La nuova classe di implementazione deve discendere direttamente dall'interfaccia richiamabile. Il registro delle chiamate, con cui si registrano le interfacce richiamabili e le loro classi di implementazione, tiene traccia delle classi implementate da ogni interfaccia registrata e la rende disponibile al componente invoker nel momento in cui l'invoker ha la necessità di chiamare l'interfaccia. Può rilevare che una classe implementa un'interfaccia solo se l'interfaccia è inclusa direttamente nella dichiarazione della classe. Non è in grado di supportare un'interfaccia nel caso sia stata ereditato insieme a una classe base.
- La nuova classe di implementazione deve includere il supporto per i metodi IUnknown che fanno parte di una qualsiasi interfaccia. Questo punto può sembrare ovvio, ma è molto facile trascurarlo. Per ulteriori informazioni su IUnknown, vedere "Creazione di classi che supportano IUnknown" a pagina 13-4.
- È necessario modificare il codice generato che registra la classe di implementazione in mod da includere un metodo factory per creare le istanze della classe di implementazione.

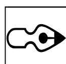
Quest'ultimo punto richiede qualche spiegazione. Quando la classe di implementazione discende da *TInvokableClass* e non sostituisce il costruttore ereditato con un nuovo costruttore che include uno o più parametri, il registro delle chiamate sa come creare le istanze della classe nel momento in cui ne ha bisogno. Quando si scrive una classe di implementazione che non discende da *TInvokableClass* o quando si modifica il costruttore, è necessario dire al registro delle chiamate in che modo ottenere le istanze della classe di implementazione.

È possibile dire al registro delle chiamate come ottenere le istanze della classe di implementazione fornendolo di una procedura factory. Anche se si ha una classe di

implementazione che discende da *TInvokableClass* e che utilizza il costruttore ereditato, è comunque possibile fornire una procedura factory. Ad esempio, invece di richiedere al registro delle chiamate di creare una nuova istanza ogni volta che l'applicazione riceve una chiamata all'interfaccia richiamabile, è possibile utilizzare una singola istanza globale della classe di implementazione.

La procedura factory deve essere di tipo *TCreateInstanceProc*. Essa restituisce un'istanza della classe di implementazione. Se la procedura crea una nuova istanza, dato che il registro delle chiamate non libera esplicitamente le istanze degli oggetti, l'oggetto di implementazione dovrebbe liberarsi da solo nel momento in cui il contatore dei riferimenti sulla sua interfaccia arriva a zero. Il codice seguente illustra un altro metodo, in la procedura factory restituisce una singola istanza globale della classe di implementazione:

```
void __fastcall CreateEncodeDecode(System::TObject* &obj)
{
  if (!FEncodeDecode)
  {
    FEncodeDecode = new TEncodeDecodeImpl();
    // save a reference to the interface so that the global instance doesn't free itself
    TEncodeDecodeImpl->QueryInterface(FEncodeDecodeInterface);
  }
  obj = FEncodeDecode;
}
```



Nell'esempio precedente, *FEncodeDecodeInterface* è una variabile di tipo *_di_IEncodeDecode*.

Si registra la procedura di fabbrica con una classe di implementazione fornendola come secondo parametro alla chiamata che registra la classe con il registro delle chiamate. Individuare Per prima cosa, si deve individuare la chiamata che il wizard ha generato per registrare la classe di implementazione. Questa apparirà nel metodo RegTypes alla fine della unit che definisce la classe. Dovrebbe assomigliare a qualcosa del genere:

```
InvRegistry()->RegisterInvokableClass(__classid(TEncodeDecodeImpl));
```

Aggiungere a questa chiamata un secondo parametro che specifichi la procedura factory:

```
InvRegistry()->RegisterInvokableClass(__classid(TEncodeDecodeImpl), &CreateEncodeDecode);
```

Utilizzo di Web Services Importer

Per utilizzare Web Services Importer, scegliere File | New | Other e alla pagina WebServices fare doppio clic sull'icona Web Services Importer. Nella finestra di dialogo visualizzata, specificare il nome di file di un documento WSDL (o di un file XML) oppure specificare l'URL in cui quel documento è pubblicato.

Se il documento WSDL è su un server che necessita di autenticazione (o deve essere raggiunto utilizzando un proxy server che richiede l'autenticazione), prima che il wizard possa acquisire il documento WSDL è necessario fornire un nome utente e una password. Per fornire queste informazioni, fare clic sul pulsante Options e fornire le opportune informazioni di connessione.

Quando si fa clic sul pulsante Next, Web Services Importer mostra il codice, compatibile con il framework di Web Services, che esso genera per ogni definizione nel documento WSDL. Ciò significa che utilizza solo quei tipi di porta che hanno un binding SOAP. Facendo clic sul pulsante Options e scegliendo le opzioni desiderate, è possibile configurare il modo in cui l'importatore genera il codice.

Web Services Importer può essere utilizzato quando si scrive un server o un'applicazione client. Quando si scrive un server, fare clic sul pulsante Options e nella finestra di dialogo risultante, attivare l'opzione che dice all'importatore di generare i codice per il server. Quando si seleziona questa opzione, l'importatore genera le classi di implementazione per le interfacce richiamabili ed è necessario solo completare i corpi dei metodi.



Se si importa un documento WSDL per creare un server che implementa un Web Service che è già stato definito, è sempre necessario pubblicare il documento WSDL per quel servizio. Fra il documento WSDL importato e l'implementazione generata vi possono essere delle differenze secondarie. Ad esempio, se il documento WSDL o file di schema XML utilizza identificativi che sono anche parole chiave del linguaggio C++, l'importatore adatta automaticamente i loro nomi in modo che il codice generato possa essere compilato.)

Non appena si fa clic su Finish, l'importatore crea nuove unit che definiscono e registrano le interfacce richiamabili per le operazioni definite nel documento, e che definiscono e registrano le classi remotable per i tipi definiti dal documento.

Come metodo alternativo, è possibile utilizzare invece l'importer WSDL per la riga comandi. Per un server, chiamare l'importatore della riga comandi con l'opzione -S, nel seguente modo:

```
WSDLIMP -S -C -V MyWSDLDoc.wsdl
```

Per un'applicazione client, chiamare l'importatore della riga comandi senza l'opzione -S:

```
WSDLIMP -C -V MyWSDLDoc.wsdl
```

Creazione di classi di eccezione custom per Web Services

Nel caso l'applicazione Web Service sollevi un'eccezione mentre sta cercando di eseguire una richiesta SOAP, essa codifica automaticamente le informazioni per quell'eccezione in un pacchetto di errore SOAP, che viene restituito al posto dei risultati della chiamata al metodo. L'applicazione client quindi solleva l'eccezione.

Per impostazione predefinta, l'applicazione client solleva un'eccezione generica di tipo *ERemotableException*with con le informazioni relative al pacchetto di errore SOAP. Se si deriva un discendente *ERemotableException*, è possibile trasmettere ulteriori informazioni specifiche per l'applicazione. I valori di tutte le proprietà pubblicate che vengono aggiunte alla classe di eccezione vengono inclusi nel pacchetto di errore SOAP in modo che il client possa sollevare un'eccezione equivalente.

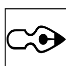
Per utilizzare un discendente di *ERemotableException* , è necessario registrarlo con il registro dei tipi remotable. Nella unit che definisce il discendente di

ERemotableException è necessario includere InvokeRegistry.hpp e aggiungere una chiamata al metodo *RegisterXSClass* dell'oggetto restituito dalla funzione globale *RemTypeRegistry*.

Se anche il client definisce e registra il discendente di *ERemotableException*, quando riceve il pacchetto di errore SOAP, solleva automaticamente un'istanza della classe di eccezione appropriata, con tutte le proprietà impostate ai valori inclusi nel pacchetto di errore SOAP.

Generazione di documenti WSDL per un'applicazione Web Service

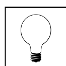
Per consentire alle applicazioni client di sapere quali sono i Web Services resi disponibili dall'applicazione, è possibile pubblicare un documento WSDL che descrive le interfacce richiamabili e indica come chiamarle.



È sempre necessario pubblicare un documento WSDL per l'applicazione Web Service, anche se si è implementato un servizio importato o se il client è stato scritto utilizzando C++Builder.

Per pubblicare un documento WSDL che descrive il Web Service, includere nel Web Module un componente *TWSDLHTMLPublish*. (SOAP Server Application wizard aggiunge questo componente automaticamente). *TWSDLHTMLPublish* è un componente auto-dispatching, il che significa che risponde automaticamente ai messaggi in arrivo che richiedono un elenco di documenti WSDL per il Web Service. Per specificare le informazioni di percorso dell'URL che i client devono utilizzare per accedere all'elenco di documenti WSDL, utilizzare la proprietà *WebDispatch*. Il Web browser può richiedere quindi l'elenco dei documenti WSDL specificando un'URL che è costituito dall'ubicazione dell'applicazione server seguita dal percorso specificato nella proprietà *WebDispatch*. L'URL assomiglia a quello mostrato di seguito:

```
http://www.myco.com/MyService.dll/WSDL
```



Se si desidera invece utilizzare un file fisico WSDL, è possibile visualizzare il documento WSDL nel Web browser e quindi salvarlo in modo da generare un file di documento WSDL.

Non è necessario pubblicare il documento WSDL dalla stessa applicazione che implementa il Web Service. Per creare un'applicazione che si limita a pubblicare il documento WSDL, omettere il codice che implementa e registra gli oggetti di implementazione e includere solo il codice che definisce e registra le interfacce richiamabili, le classi remotable che rappresentano i tipi complessi e le eventuali eccezioni remotable.

Per impostazione predefinita, quando si pubblica un documento WSDL, esso indica che i servizi sono disponibili allo stesso URL su cui è stato pubblicato il documento WSDL (anche se con un percorso diffrente). Se si stanno distribuendo più versioni dell'applicazione Web Service oppure se si pubblica il documento WSDL da un'applicazione diversa da quella che implementa il Web Service, sarà necessario modificare il documento WSDL in modo che includa le informazioni aggiornate su dove individuare il Web Service.

Per modificare l'URL, utilizzare il WSDL administrator. La prima operazione consiste nell'attivare l'amministratore. L'operazione viene condotta impostando la proprietà *AdminEnabled* del componente *TWSDLHTMLPublish* a **true**. In seguito, quando si utilizzerà il browser per visualizzato l'elenco dei documenti WSDL, esso includerà anche un pulsante che ne consentirà la gestione. Utilizzare il WSDL administrator per specificare le ubicazioni (URL) su cui è stata distribuita l'applicazione Web Service.

Scrittura di client per Web Services

C++Builder fornisce un supporto sul lato client per chiamare i Web Services che utilizzano un binding basato su SOAP. Questi Web Services possono essere forniti da un server scritto in C++Builder o da un qualsiasi altro server che definisce il proprio Web Service mediante un documento WSDL.

Importazione di documenti WSDL

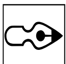
Prima di potere utilizzare un Web Service, l'applicazione deve definire e registrare le interfacce e i tipi richiamabili inclusi nell'applicazione Web Service. Per ottenere queste definizioni, è possibile importare un documento WSDL (o un file XML) che definisce il servizio. Web Services crea una unit che definisce e registra le interfacce e i tipi che è necessario utilizzare. Per i dettagli sull'utilizzo di Web Services Importer, vedere "Utilizzo di Web Services Importer" a pagina 36-14.

Chiamata a interfacce richiamabili

Per chiamare un'interfaccia richiamabile, l'applicazione client deve includere tutti gli header che definiscono le interfacce richiamabili e tutte le classi remotable che implementano tipi complessi.

Una volta che l'applicazione client ha la dichiarazione di un'interfaccia richiamabile, creare un'istanza di *THTTPRio* per l'interfaccia desiderata:

```
X = new THTTPRio(NULL);
```



È importante che non si distrugga esplicitamente l'istanza di *THTTPRio*. Se è stato creato senza un *Owner* (come nella riga di codice vista in precedenza), esso si libera automaticamente non appena viene rilasciata la sua interfaccia. Se è stato creato con un *Owner*, è l'*Owner* che si fa carico di liberare l'istanza di *THTTPRio*.

Quindi, fornire all'oggetto *THTTPRio* le informazioni necessarie per identificare l'interfaccia del server e individuare il server. Vi sono due modi per fornire queste informazioni:

- Se non si fa affidamento sull'URL per il Web Service o sui namespace e sugli header Soap che l'oggetto deve modificare, è possibile specificare semplicemente l'URL per il Web Service a cui si vuole accedere. *THTTPRio* utilizza questo URL per cercare la definizione dell'interfaccia, oltre a tutte le informazioni sui namespace e sgli header, in base alle informazioni contenute nel registro delle

chiamate. Specificare l'URL impostando la proprietà *URL* all'ubicazione del server:

```
X->URL = "http://www.myco.com/MyService.dll/SOAP/IServerInterface";
```

- Se si desidera cercare dinamicamente in fase di esecuzione l'URL, il namespace o il Soap Action header nel documento WSDL, è possibile utilizzare le proprietà *WSDLLocation*, *Service* e *Port* e l'oggetto estrarrà le informazioni necessarie dal documento WSDL:

```
X.WSDLLocation = "Cryptography.wsdl";
X.Service = "Cryptography";
X.Port = "SoapEncodeDecode";
```

Una volta che si è specificato come individuare il server e identificare l'interfaccia, è possibile utilizzare il metodo *QueryInterface* per ottenere un puntatore all'interfaccia per l'interfaccia richiamabile. In questo modo viene creata dinamicamente in memoria una vtable per l'interfaccia associata, consentendo così di effettuare chiamate all'interfaccia:

```
_di_IEncodeDecode InterfaceVariable;
X->QueryInterface(InterfaceVariable);
if (InterfaceVariable)
{
  Code = InterfaceVariable->EncodeValue(5);
}
```

Notare che la chiamata a *QueryInterface* accetta come argomento il wrapper *DelphiInterface* per l'interfaccia richiamabile al posto della stessa interfaccia richiamabile.

THTTPRio si basa sul registro delle chiamate per ottenere le informazioni relative all'interfaccia richiamabile. Se l'applicazione client non ha un registro delle chiamate o se l'interfaccia richiamabile non è registrata, *THTTPRio* non può costruire la propria vtable in memoria.



Se si assegna a una variabile globale l'interfaccia che si ottiene da *THTTPRio*, prima di chiudere l'applicazione è necessario modificare quell'assegnazione a NULL. Ad esempio, se *InterfaceVariable* nell'esempio di codice precedente fosse una variabile globale invece che una variabile di stack, sarebbe necessario rilasciare l'interfaccia prima che l'oggetto *THTTPRio* venga liberato. Di solito, questo codice va nel gestore di evento *OnDestroy* della scheda o del modulo dati:

```
void __fastcall TForm1::FormDestroy(TObject *Sender)
{
  InterfaceVariable = NULL;
}
```

Il motivo per cui è necessario riassegnare a una variabile di interfaccia globale il valore NULL sta nel fatto che *THTTPRio* costruisce dinamicamente in memoria la propria vtable. Quella vtable deve essere ancora presente quando l'interfaccia viene rilasciata. Se non si rilascia l'interfaccia insieme alla scheda o al modulo dati, essa viene rilasciata quando la variabile globale viene liberata al momento della chiusura. La memoria per le variabili globali può essere liberata dopo la scheda o il modulo dati che contiene l'oggetto *THTTPRio*, nel qual caso la vtable non sarà disponibile quando viene rilasciata l'interfaccia.

Capitolo

37

Operazioni con i socket

Questo capitolo descrive i componenti socket che consentono di creare un'applicazione che, grazie a TCP/IP e ai relativi protocolli, è in grado di comunicare con altri sistemi. Grazie ai socket, è possibile leggere e scrivere su altre macchine connesse senza preoccuparsi dei dettagli del software di rete sottostante. I socket forniscono connessioni basate sul protocollo TCP/IP, ma sono sufficientemente generici da operare con protocolli analoghi quali User Datagram Protocol (UDP), Xerox Network System (XNS), DECnet di Digital, o la famiglia IPX/SPX di Novell.

L'utilizzo dei socket consente di scrivere applicazioni server o client di rete che leggono e scrivono su altri sistemi. Un'applicazione server o client è solitamente dedicata a un singolo servizio, quale Hypertext Transfer Protocol (HTTP) o File Transfer Protocol (FTP). Grazie ai socket server, un'applicazione in grado di fornire uno di questi servizi può effettuare un collegamento con applicazioni client che vogliono usare quel servizio. I socket client permettono di collegare un'applicazione che usa uno di questi servizi alle applicazioni server che lo forniscono.

Implementazione dei servizi

I socket forniscono uno degli elementi necessari per scrivere applicazioni server o client di rete. Per molti servizi, quali ad esempio HTTP o FTP, sono già disponibili server di terze parti. Alcuni sono addirittura incorporati nel sistema operativo, per cui non occorre nemmeno scriverli. Comunque, quando si vuole un maggior controllo sul modo in cui il servizio viene implementato, o un'integrazione più stretta tra la propria applicazione e la comunicazione di rete, o quando non è disponibile alcun server per quel determinato servizio di cui si ha bisogno, è possibile creare un'applicazione server o client personalizzata. Per esempio, quando si opera con dataset distribuiti, è possibile scrivere uno strato per comunicare con database posti su altri sistemi.

I protocolli di servizio

Prima di poter scrivere un server o un client di rete, occorre capire il servizio che l'applicazione deve fornire o utilizzare. Molti servizi hanno protocolli standard che devono essere supportati dall'applicazione di rete. Se si sta scrivendo un'applicazione di rete per un servizio standard quale, ad esempio, HTTP, FTP, o anche Finger o Time, occorre anzitutto capire i protocolli usati per comunicare con gli altri sistemi. Consultare la documentazione su un determinato servizio che viene fornito o utilizzato.

Se si sta fornendo un nuovo servizio per un'applicazione che comunica con altri sistemi, il primo passo consiste nella progettazione del protocollo di comunicazione per i server e i client di questo servizio. Quali messaggi vengono trasmessi? Come vengono coordinati questi messaggi? Come sono codificate le informazioni?

Comunicare con le applicazioni

Spesso, l'applicazione server o client fornisce uno strato tra il software di rete e un'applicazione che usa il servizio. Un server HTTP, ad esempio, risiede tra Internet e un'applicazione server web che fornisce le pagine di contenuti e risponde ai messaggi di richiesta HTTP.

I socket forniscono l'interfaccia tra l'applicazione server o client e il software di rete. Occorre fornire l'interfaccia tra la propria applicazione e i client che la utilizzano. È possibile copiare l'API di un server standard di terze parti (come ad esempio Apache) oppure si può progettare e rendere pubblica la propria API.

Servizi e porte

La maggior parte dei servizi standard sono associati, per convenzione, con determinati numeri di porta. Parleremo dettagliatamente dei numeri di porta nelle pagine seguenti. Per il momento, si consideri il numero di porta come un codice numerico per il servizio.

Se si sta implementando un servizio standard in applicazioni multipiattaforma, gli oggetti socket di Linux mettono a disposizione dei metodi che consentono di cercare il numero di porta del servizio. Se si sta fornendo un nuovo servizio, è possibile specificare il numero di porta associato nel file /etc/services. Per maggiori informazioni sul file services, consultare la documentazione di Linux.

Tipi di connessioni socket

Le connessioni socket possono essere divise in tre tipi fondamentali che indicano il modo in cui si è iniziata la connessione e a quale socket locale è connesso. Questi tipi sono

- Connessioni client.
- Connessioni di ascolto.

- Connessioni server.

Una volta stabilita una connessione con un socket client, la connessione server è praticamente indistinguibile da una connessione client. Entrambi i punti terminali hanno le stesse funzionalità e ricevono gli stessi tipi di evento. Solo la connessione di ascolto è fondamentalmente differente, in quanto possiede soltanto un singolo punto terminale.

Connessioni client

Le connessioni client connettono un socket client posto sul sistema locale con un socket server posto su un sistema remoto. Le connessioni client vengono iniziate dal socket client. Per prima cosa, il socket client deve descrivere il socket server a cui desidera connettersi. Il socket client quindi cerca il socket server e, una volta localizzatolo, richiede una connessione. Il socket server può non completare la connessione immediatamente. I socket server conservano una coda di richieste client e completano le connessioni quando ne hanno il tempo. Quando il socket server accetta la connessione client, trasmette al socket client una descrizione completa del socket server al quale si sta connettendo e la connessione viene completata dal client.

Connessioni di ascolto

I socket server non localizzano i client. Formano invece “mezze connessioni” passive che ascoltano le richieste del client. I socket server associano una coda alle loro connessioni di ascolto; la coda registra le richieste di connessione client via via che queste vengono inoltrate. Quando il socket server accetta una richiesta di connessione dall’applicazione client, forma un nuovo socket per connettersi con client, così da lasciare la connessione di ascolto aperta per accettare altre richieste client.

Connessioni server

Nel momento in cui il socket di ascolto accetta una richiesta del client, il socket server forma le connessioni server. Quando il server accetta la connessione, il client riceve una descrizione del socket server che completa la connessione. La connessione viene stabilita quando il socket client riceve questa descrizione e completa la connessione.

Descrizione dei socket

I socket consentono a un’applicazione per rete di comunicare con gli altri sistemi della rete. Ciascun socket può essere considerato come un punto terminale di una connessione in rete. Dispone di un indirizzo che specifica:

- Il sistema su cui è in esecuzione.
- I tipi di interfacce che è in grado di capire.

- La porta utilizzata per la connessione.

Per descrivere appieno una connessione socket occorre fornire gli indirizzi dei socket su entrambe le estremità della connessione. È possibile descrivere l'indirizzo di ciascun punto terminale del socket o indicandone l'indirizzo IP oppure l'host e il numero della porta.

Prima di poter stabilire una connessione socket, è necessario descrivere completamente i socket costituenti i relativi punti terminali. Alcune informazioni sono reperibili dal sistema su cui è in esecuzione l'applicazione. Non è ad esempio necessario descrivere l'indirizzo IP locale di un socket client dato che questa informazione è disponibile da sistema operativo.

Le informazioni che occorre fornire dipendono dal tipo di socket col quale si sta operando. I socket client devono descrivere il server con il quale vogliono connettersi. I socket server di ascolto devono descrivere la porta che rappresenta il servizio che forniscono.

Descrizione dell'host

L'host è il sistema che sta eseguendo l'applicazione contenente il socket. È possibile descrivere l'host di un socket fornendo il suo indirizzo IP, che è una stringa di quattro valori numerici (byte) nella notazione con punti standard di Internet, come ad esempio

```
123.197.1.2
```

Un singolo sistema può supportare più di un indirizzo IP.

Gli indirizzi IP sono spesso difficili da ricordare e capita spesso di scriverli in modo non corretto. Un'alternativa è quella di usare il nome dell'host. I nomi di host sono alias dell'indirizzo IP che si vedono spesso negli Uniform Resource Locators (URL). Sono stringhe che contengono un nome di dominio e un servizio, quale ad esempio

```
http://www.ASite.com
```

La maggior parte delle Intranet forniscono nomi host per gli indirizzi IP dei sistemi in Internet. È possibile apprendere il nome dell'host associato a un qualsiasi indirizzo IP (nel caso esista) impartendo il seguente comando dalla riga comandi:

```
nslookup IPADDRESS
```

in cui *IPADDRESS* è l'indirizzo IP a cui si è interessati. Se al proprio indirizzo IP locale non è associato un nome di host e si vuole averne uno, si deve contattare l'amministratore della rete. È comune per i computer fare riferimento a sé stessi con il nome di *localhost* e con il numero IP 127.0.0.1.

I socket server non hanno bisogno di specificare un host. L'indirizzo IP locale può essere letto dal sistema. Se il sistema locale supporta più di un indirizzo IP, i socket server ascolteranno le richieste client simultaneamente su tutti gli indirizzi IP. Quando un socket server accetta una connessione, il socket client fornisce l'indirizzo IP remoto.

I socket client devono specificare l'host remoto fornendone il nome oppure indicando l'indirizzo IP.

Scelta tra un nome di host e un indirizzo IP

La maggior parte delle applicazioni usa il nome dell'host per specificare un sistema. I nomi di host sono più facili da ricordare ed è più facile controllare che siano stati scritti correttamente. Inoltre, i server possono modificare il sistema o l'indirizzo IP associati ad un particolare nome di host. L'uso di un nome di host permette al socket client di trovare il sito astratto rappresentato dal nome di host, anche quando è stato spostato ad un nuovo indirizzo IP.

Se il nome di host è sconosciuto, il socket client deve specificare il sistema del server usando l'indirizzo IP. Specificare il sistema del server fornendo l'indirizzo IP consente di risparmiare tempo. Quando si fornisce il nome di host, prima di poter localizzare il sistema del server, il socket deve cercare l'indirizzo IP associato a quel nome.

Uso delle porte

Benché l'indirizzo IP fornisca informazioni sufficienti per trovare il sistema all'altro capo di una connessione socket, occorre comunque anche il numero di porta di quel sistema. Senza numeri di porta, un sistema potrebbe formare solo una singola connessione alla volta. I numeri di porta sono identificatori unici che consentono a un singolo sistema di ospitare simultaneamente più connessioni, assegnando un numero di porta separato a ciascuna connessione.

In precedenza abbiamo descritto i numeri di porta come codici numerici dei servizi implementati da applicazioni in rete. Questa, in realtà, è solo una convenzione che permette alle connessioni di ascolto server di rendersi disponibili su un numero di porta fisso in modo da poter essere trovate dai socket client. I socket server ascoltano le richieste sul numero di porta associato al servizio che forniscono. Quando accettano una connessione con un socket client, creano una connessione socket separata che usa un numero di porta diverso e arbitrario. In questo modo, la connessione di ascolto può continuare ad ascoltare le richieste sul numero di porta associato al servizio.

I socket client usano un numero di porta locale arbitrario perché non devono essere trovati da nessun altro socket. Specificano il numero di porta del socket server col quale vogliono connettersi in modo da poter trovare l'applicazione server. Spesso, questo numero di porta viene specificato indirettamente chiamando il servizio desiderato.

Utilizzo dei componenti socket

La pagina Internet della Component palette include tre componenti socket che permettono a un'applicazione di rete di formare connessioni con altre macchine e all'utente di leggere e scrivere informazioni su quella connessione. Questi componenti sono:

- *TcpServer*
- *TcpClient*

- *UdpSocket*

Associati con ciascuno di questi componenti socket ci sono gli oggetti socket, che rappresentano il punto terminale di una connessione socket in uso. I componenti socket usano gli oggetti socket per incapsulare le chiamate del socket server; pertanto l'applicazione non deve preoccuparsi dei dettagli necessari per stabilire la connessione o di gestire i messaggi del socket.

Se si vogliono personalizzare i dettagli delle connessioni che i componenti socket creano per conto dell'utente, si possono usare le proprietà, gli eventi e i metodi degli oggetti socket.

Ottenimento di informazioni sulla connessione

Dopo avere completato la connessione con un socket client o server, per ottenere informazioni sulla connessione è possibile usare l'oggetto socket client o server associato al proprio componente socket. Usare le proprietà *LocalHost* e *LocalPort* per determinare l'indirizzo e il numero di porta usati dal socket client o server locale oppure usare le proprietà *RemoteHost* e *RemotePort* per determinare l'indirizzo e il numero di porta usati dal socket client o server remoto. Usare il metodo *GetSocketAddr* per generare un indirizzo di socket valido in base al nome dell'host e al numero di porta. È possibile utilizzare il metodi *LookupPort* per individuare il numero di porta. Utilizzare il metodo *LookupProtocol* method to look up the protocol number. Use the *LookupHostName* per individuare il nome dell'hosto in base all'indirizzo IP della macchina host.

Per controllare il traffico di rete in ingresso e in uscita dal socket utilizzare le proprietà *BytesSent* e *BytesReceived*.

Utilizzo di socket client

Per convertire l'applicazione in un client TCP/IP o UDP, occorre aggiungere un componente *TcpClient* o *UdpSocket* alla scheda o al modulo dati. I socket client permettono di specificare il socket server al quale ci si vuole connettere e il servizio che si vuole dal server. Una volta descritta la connessione desiderata, si può usare il componente socket client per completare la connessione con il server.

Ogni componente socket client usa un singolo oggetto socket client per rappresentare il punto terminale client di una connessione.

Specifica del server desiderato

I componenti socket client hanno alcune proprietà che permettono di specificare il sistema e la porta server a cui ci si vuole connettere. Utilizzare la proprietà *RemoteHost* per specificare il server host remoto in base al nome host o all'indirizzo IP.

Oltre al sistema server, si deve specificare la porta del sistema server con il quale il socket client si connetterà.È possibile utilizzare la proprietà *RemotePort* per

specificare il numero di porta del server in modo diretto oppure indirettamente indicando il nome del servizio di destinazione.

Formazione della connessione

Una volta impostate le proprietà del componente socket client in modo che descrivano il server con il quale ci si vuole connettere, è possibile formare la connessione durante l'esecuzione chiamando il metodo *Open*. Se si vuole che l'applicazione esegua automaticamente la connessione al momento dell'avvio, occorre, in fase di progetto, impostare la proprietà *Active* a **true**, usando l'Object Inspector.

Ottenimento di informazioni sulla connessione

Dopo avere completato la connessione con un socket server, per ottenere informazioni sulla connessione è possibile usare l'oggetto socket associato col componente socket client. Usare le proprietà *LocalHost* e *LocalPort* per determinare l'indirizzo e il numero di porta usati dai socket client e server per formare il punto terminale della connessione. Si può usare la proprietà *Handle* per ottenere un handle alla connessione socket da usare con le chiamate socket.

Chiusura della connessione

Una volta terminato la comunicazione con un'applicazione server attraverso la connessione socket, si può chiudere la connessione chiamando il metodo *Close*. La connessione può essere chiusa anche da parte del server. Se così è, l'utente ne riceverà notifica in un evento *OnDisconnect*.

Uso dei socket server

Per convertire un'applicazione in un server IP occorre aggiungere un componente socket server (*TcpServer* o *UdpSocket*) alla scheda o al modulo dati. I socket server permettono di specificare il servizio che si sta fornendo o la porta che si vuole usare per ascoltare le richieste di client. Si può usare il componente socket server per ascoltare e accettare le richieste di connessione provenienti da client.

Ogni componente socket server usa un singolo oggetto socket server per rappresentare il punto terminale server in una connessione di ascolto. Usa anche un oggetto socket server client per il punto terminale server di ciascuna connessione attiva, accettata dal server, verso un socket client.

Specifica della porta

Prima che il socket server possa ascoltare le richieste provenienti da client, si deve specificare la porta dal quale il server le ascolterà. Si può specificare questo porta usando la proprietà *LocalPort*. Se l'applicazione server corrente fornisce un servizio standard che, per convenzione, è associato a un numero di porta specifico, è anche possibile specificare il nome del servizio usando la proprietà *LocalPort*. È preferibile utilizzare il nome del servizio invece del numero di porta, perché è più facile commettere errori di battitura quando si specifica il numero di porta.

Ascolto delle richieste provenienti da client

Una volta impostato il numero di porta del componente socket del server, durante l'esecuzione è possibile formare una connessione di ascolto chiamando il metodo *Open*. Se si vuole che l'applicazione in uso esegua automaticamente la connessione di ascolto all'avvio, occorre, in fase di progetto, impostare la proprietà *Active* a **true**, usando l'Object Inspector.

Connessione con i client

Un socket server di ascolto accetta automaticamente le richieste di connessione provenienti da client, man mano che le riceve. Ogni volta che questo si verifica se ne ha notifica in un evento *OnAccept*.

Chiusura delle connessioni server

Quando si vuole chiudere la connessione di ascolto, occorre chiamare il metodo *Close* o impostare la proprietà *Active* a **false**. Questo chiude tutte le connessioni con applicazioni client aperte, annulla tutte le connessioni non accettate in corso, e poi chiude la connessione di ascolto in modo che il componente socket del server non possa più accettare alcuna nuova connessione.

Quando i client TCP chiudono le proprie singole connessioni al socket del server, si viene informati mediante un evento *OnDisconnect*.

Risposta agli eventi socket

Quando si scrivono applicazioni che utilizzano socket, è possibile scrivere sul socket o leggere dal socket da un punto qualsiasi del programma. È possibile scrivere sul socket utilizzando nel programma i metodi *SendBuf*, *SendStream*, o *Sendln* dopo che il socket è stato aperto. È possibile leggere dal socket utilizzando metodi che hanno un nome analogo *ReceiveBuf* e *Receiveln*. Gli eventi *OnSend* e *OnReceive* vengono innescati ogni volta che si scrive o si legge qualcosa dal socket. Possono essere utilizzati per operazioni di filtro. Ogni volta che si legge o si scrive, viene innescato un evento di lettura o di scrittura.

I socket client e i socket server generano entrambi eventi di errore quando ricevono messaggi di errore dalla connessione.

I componenti socket ricevono inoltre due eventi durante l'apertura e il completamento di una connessione. Se l'applicazione ha l'esigenza di influenzare il modo in cui avviene l'apertura del socket è necessario usare i metodi *SendBuf* e *ReceiveBuf* per rispondere a questi eventi del client o del server.

Eventi di errore

I socket client e server generano eventi di errore quando ricevono messaggi di errore dalla connessione. È possibile scrivere un gestore di evento *OnError* che risponda a questi messaggi di errore. Al gestore di evento vengono passate informazioni relativamente a

- Quale oggetto socket ha ricevuto la notifica dell'errore.
- Ciò che il socket stava cercando di fare quando si è verificato l'errore.
- Il codice di errore fornito dal messaggio di errore.

È possibile rispondere all'errore nel gestore di evento e cambiare il codice di errore in 0 per impedire al socket di sollevare un'eccezione.

Eventi del client

Quando un socket client apre una connessione, si verificano i seguenti eventi:

- Viene impostato e inizializzato il socket per la notifica dell'evento.
- Si verifica un evento *OnCreateHandle* dopo che il server e il socket server sono stati individuati. A questo punto, l'oggetto socket disponibile tramite la proprietà *Handle* può fornire informazioni sul socket server o client che costituirà l'altra estremità della connessione. Questa è la prima possibilità per ottenere la porta effettivamente usata per la connessione, che può differire dalla porta del socket di ascolto che ha accettato la connessione.
- La richiesta di connessione viene accettata dal server e completata dal socket client.
- Una volta stabilita la connessione, si verifica l'evento di notifica *OnConnect*.

Eventi del server

I componenti socket server permettono di stabilire due tipi di connessione: connessioni di ascolto e connessioni alle applicazioni client. Il socket server riceve eventi durante la formazione di ciascuna di queste connessioni.

Eventi durante l'ascolto

Poco prima che venga formata la connessione di ascolto, si verifica l'evento *OnListening*. È possibile usare la sua proprietà *Handle* per apportare modifiche al socket prima che venga aperto per l'ascolto. Per esempio, se si vogliono limitare gli indirizzi IP che il server usa per l'ascolto, occorre farlo in un gestore di evento *OnListening*.

Eventi con connessioni client

Quando un socket server accetta una richiesta di connessione del client, si verificano i seguenti eventi:

- Si verifica un evento *OnAccept* che passa il nuovo oggetto *TTcpClient* al gestore dell'evento. Questa è la prima occasione in cui è possibile usare le proprietà di *TTcpClient* per ottenere informazioni sul punto terminale server della connessione ad un client.
- Se *BlockMode* è *bmThreadBlocking* si verifica un evento *OnGetThread*. Se si desidera utilizzare un proprio discendente custom di *TServerSocketThread*, è possibile

crearne uno in un gestore di evento OnGetThreaded esso sarà utilizzato al posto di *TServerSocketThread*. Se si desidera eseguire una inizializzazione del thread, o effettuare una qualsiasi chiamata API al socket prima che il thread inizi a leggere o scrivere sulla connessione, usare anche in questi casi il gestore di evento *OnGetThread*.

- Il client completa la connessione e si verifica un evento *OnAccept*. Con un server non bloccante può essere opportuno a questo punto iniziare la lettura o la scrittura sulla connessione socket

Lettura e scrittura sulle connessioni socket

La ragione per cui si stabiliscono connessioni socket con altre macchine è che in questo modo si ha la possibilità di leggere o scrivere informazioni su quelle connessioni. Il tipo di informazioni da leggere o da scrivere, o quando farlo, dipendono dal servizio associato alla connessione socket.

La lettura e la scrittura sui socket possono verificarsi in modo asincrono, così da non bloccare l'esecuzione di altro codice sull'applicazione in rete. Questa connessione è chiamata non bloccante. È anche possibile formare connessioni bloccanti, in cui un'applicazione aspetta il termine della lettura o della scrittura prima di eseguire la linea di codice successiva.

Connessioni non bloccanti

Le connessioni non bloccanti leggono e scrivono in modo asincrono, in modo che il trasferimento di dati non blocchi l'esecuzione di altro codice nell'applicazione in rete. Per crere una connessione non bloccante per socket client o server, impostare la proprietà *BlockMode* a *bmNonBlocking.*

Quando la connessione è non bloccante, gli eventi di lettura e scrittura informano il socket quando il socket all'altra estremità della connessione tenta di leggere o scrivere informazioni.

Eventi di lettura e scrittura

I socket non bloccanti generano eventi di lettura e scrittura quando tentano di leggere o scrivere sulla connessione. È possibile rispondere a queste notifiche nei gestori di evento *OnReceive* o *OnSend*.

L'oggetto socket associato alla connessione socket viene fornito sotto forma di parametro ai gestori di evento di lettura e scrittura. Questo oggetto socket possiede una serie di metodi che consentono di leggere o di scrivere sulla connessione.

Per leggere dalla connessione socket, utilizzare i metodi ReceiveBufo *Receiveln*. Per scrivere sulla connessione socket, utilizzare i metodi *SendBuf*, *SendStream*, o *Sendln*.

Connessioni bloccanti

Quando la connessione è di tipo bloccante, il proprio socket deve iniziare a leggere o a scrivere sulla connessione. Il socket non può aspettare passivamente una notifica dalla connessione. Si usi un socket di tipo bloccante quando la propria estermità della connessione ha la responsabilità del momento in cui si verificano operazioni di lettura o di scrittura.

Per socket client o server, impostare la proprietà *BlockMode* a *bmBlocking* per formare una connessione di tipo bloccante.A seconda di ciò che fa l'applicazione client, è possibile creare un nuovo thread esecutivo per leggere o scrivere, in modo da permettere all'applicazione di continuare ad eseguire il codice su altri thread mentre aspetta che la lettura o la scrittura sulla connessione vengano completate.

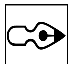
Per i socket server, impostare la proprietà *BlockMode* a *bmBlocking* o *bmThreadBlocking* per formare una connessione di tipo bloccante. Dato che, mentre il socket aspetta le informazioni da scrivere o leggere sulla connessione, le connessioni di tipo bloccante interrompono l'esecuzione di tutto il resto del codice, quando il *BlockMode* è *bmThreadBlocking* i componenti socket server generano sempre un nuovo thread esecutivo per ogni connessione client. Se *BlockMode* è impostata a *bmBlocking* l'esecuzione di programma viene bloccata finché non si stabilisce una nuova connessione.

Parte

IV

Sviluppo di applicazioni COM

I capitoli inclusi nella sezione "Sviluppo di applicazioni COM" presentano i concetti necessari per generare applicazioni basate su COM, fra cui i controller Automation, i server Automation, i controlli ActiveX e le applicazioni COM+.



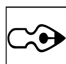
Il supporto per i client COM è disponibile in tutte le versioni di C++Builder. Tuttavia, per creare i server, bisogna disporre della versione Professional o Enterprise.

Capitolo

38

Panoramica sulle tecnologie COM

C++Builder fornisce wizard e classi che facilitano l'implementazione di applicazioni basate sul Component Object Model (COM) di Microsoft. Con questi wizard è possibile creare classi basate su COM e componenti da usare all'interno di applicazioni, oppure client o server COM che implementano oggetti COM, server Automation (inclusi gli Active Server Objects), controlli ActiveX o ActiveForms completamente funzionali.



I componenti COM, come quelli sulle pagine ActiveX, COM+ e Servers della Component palette, non possono essere utilizzati nelle applicazioni CLX. Questa tecnologia è destinata al solo utilizzo in ambiente Windows e non è multipiattaforma.

COM è un modello software a componenti, indipendente dal linguaggio, che consente l'interazione tra i componenti software e le applicazioni eseguiti su piattaforma Windows. L'aspetto chiave di COM è la sua capacità di consentire la comunicazione fra componenti, fra applicazioni e fra client e server tramite interfacce chiaramente definite. Le interfacce consentono ai client di chiedere a un componente COM quali funzioni supporta in esecuzione. Per dotare il componente di ulteriori funzioni, è sufficiente aggiungere una opportuna interfaccia.

Le applicazioni possono accedere alle interfacce dei componenti COM, presenti sullo stesso computer su cui risiede l'applicazione o che esistono su un altro computer della rete, mediante un meccanismo detto Distributed COM (DCOM). Per maggiori informazioni sui client, sui server e sulle interfacce, consultare la sezione, "Parti di un'applicazione COM", a pagina 38-3.

Questo capitolo fornisce una panoramica concettuale della tecnologia sulla quale i controlli Automation e ActiveX sono costruiti. Nei capitoli successivi vengono riportate informazioni dettagliate sulla creazione degli oggetti Automation e dei controlli ActiveX in C++Builder.

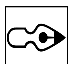
COM come specifica e implementazione

COM è al contempo una specifica e un'implementazione. Le specifiche COM definiscono le modalità per la creazione di oggetti e le modalità di comunicazione tra

oggetti. In base a tali specifiche, gli oggetti COM possono essere scritti i diversi linguaggi, funzionare in spazi di elaborazione differenti e su piattaforme diverse. Finché gli oggetti sono conformi alla specifica scritta, possono comunicare. Ciò consente di integrare il codice preesistente, relativo per esempio a un componente, con nuovi componenti implementati nei linguaggi orientati agli oggetti.

L'implementazione COM è inclusa nella libreria Win32, la quale fornisce numerosi servizi importanti che supportano le specifiche scritte. La libreria COM contiene una serie di interfacce standard, che definiscono la funzionalità centrale di un oggetto COM, e un piccolo set di funzioni API, progettate specificatamente per la creazione e la gestione degli oggetti COM.

Quando in un'applicazione si usano i wizard e gli oggetti VCL di C++Builder, si utilizza l'implementazione di C++Builder per la specifica COM. Inoltre, C++Builder fornisce alcuni wrapper per i servizi COM per quelle funzionalità che non implementa direttamente (come gli Active Document).



C++Builder implementa gli oggetti in base alle specifiche COM utilizzando la Microsoft Active Template Library (ATL), modificata per classi e macro.

Estensioni COM

Essendosi evoluto, COM si è esteso oltre i servizi COM. COM è la base di altre tecnologie, come Automation, controlli ActiveX, Active Documents e Active Directories. Per informazioni dettagliate sulle estensioni COM, consultare "Estensioni COM" a pagina 38-10.

Inoltre, quando si lavora all'interno di vasti ambienti distribuiti, è possibile creare oggetti COM transazionali. Prima di Windows 2000, tali oggetti non facevano parte dell'architettura COM, ma piuttosto pensati per funzionare in ambiente Microsoft Transaction Server (MTS). Con l'avvento di Windows 2000, tale supporto è stato integrato in COM+. Gli oggetti transazionali sono descritti in dettaglio nel Capitolo 44, "Creazione di oggetti MTS o COM+".

C++Builder fornisce i wizard per implementare con facilità applicazioni che incorporano le tecnologie sopra menzionate. Per informazioni dettagliate, consultare il paragrafo "Implementazione di oggetti COM con i wizard" a pagina 38-20.

Parti di un'applicazione COM

Quando si implementa un'applicazione COM, occorre fornire:

Interfaccia COM Il modo in cui un oggetto espone i suoi servizi all'esterno dei client. Un oggetto COM fornisce un'interfaccia per ciascun gruppo di metodi correlati e proprietà. Si noti che le proprietà COM non sono identiche alle proprietà degli oggetti della VCL. Le proprietà COM utilizzano sempre metodi di accesso in lettura e scrittura e non vengono dichiara utilizzando la parola chiave **__property**.

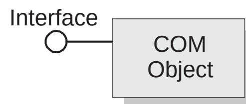
Server COM Un modulo (EXE, DLL o OCX) che contiene il codice per un oggetto COM. Le implementazioni dell'oggetto risiedono nei server. Un oggetto COM implementa una o più interfacce.

Client COM Il codice che chiama le interfacce per ottenere i servizi richiesti dal server. I client sanno che cosa acquisire dal server (tramite l'interfaccia), ma non sanno come il server fornisce i servizi. C++Builder semplifica il processo di creazione di un client, consentendo l'installazione di server COM (come un documento Word o una presentazione PowerPoint) sotto forma di componenti. Ciò consente di effettuare la connessione al server e di agganciarne gli eventi mediante l'Object Inspector.

Interfacce COM

I client COM comunicano con gli oggetti tramite interfacce COM. Le interfacce sono gruppi di routine correlate logicamente o semanticamente, che forniscono la comunicazione tra un provider di un servizio (oggetto server) e i suoi client. Il modo standard per illustrare un'interfaccia COM è mostrato nella Figura 38.1:

Figura 38.1 Un'interfaccia COM



Ad esempio, ogni oggetto COM deve implementare l'interfaccia base, *IUnknown*. Mediante una routine di nome *QueryInterface* in *IUnknown*, i client possono richiedere altre interfacce implementate dal server.

Gli oggetti possono avere più interfacce, ciascuna delle quali implementa una funzionalità diversa. Un'interfaccia fornisce un modo per trasmettere al client quale servizio fornisce, senza fornire i particolari dell'implementazione, ovvero come o dove l'oggetto fornisce questo servizio.

Gli aspetti fondamentali delle interfacce COM sono i seguenti:

- Una volta pubblicate, le interfacce sono immutabili, cioè non cambiano. È possibile fare affidamento su un'interfaccia per fornire uno specifico set di funzioni. Un'ulteriore funzionalità viene fornita da altre interfacce.

- Per convenzione, gli identificatori dell'interfaccia COM iniziano con una lettera maiuscola I, seguita poi da un nome simbolico che definisce l'interfaccia, come *IMalloc* o *IPersist*.
- Le interfacce hanno un'identificazione univoca, chiamata **Globally Unique Identifier (GUID)**, che è un numero a 128 bit generato in modo casuale. I GUID dell'interfaccia vengono chiamati **Interface Identifiers (IID)**. Ciò elimina i conflitti nei nomi tra versioni differenti di un prodotto o tra prodotti diversi.
- Le interfacce sono indipendenti dal linguaggio. È possibile usare qualsiasi linguaggio per implementare un'interfaccia COM a condizione che supporti una struttura di puntatori e possa chiamare, in modo esplicito o implicito, una funzione tramite un puntatore.
- Le interfacce non sono di per sé oggetti; esse forniscono un modo per accedere ad un oggetto. Pertanto, i client non accedono direttamente ai dati, ma attraverso un puntatore di interfaccia. Windows 2000 aggiunge un'ulteriore strato di indirezione, noto come interceptor, grazie a cui fornisce funzioni COM+ come l'attivazione just-in-time e la centralizzazione degli oggetti.
- Le interfacce sono sempre ereditate dall'interfaccia fondamentale, *IUnknown*.
- Le interfacce possono essere reindirizzate da COM tramite i proxy per consentire chiamate al metodo dell'interfaccia per la comunicazione tra thread, processi e macchine collegate in rete, il tutto senza che gli oggetti client o server debbano preoccuparsi del reindirizzamento. Per ulteriori informazioni consultare il paragrafo , "Server in-process, out-of-process e remoti", a pagina 38-6.

L'interfaccia COM fondamentale, IUnknown

l Tutti gli oggetti COM devono supportare l'interfaccia fondamentale, *IUnknown*, *un* ***typedef*** *per il tipo interfacia di base IInterface*. *IUnknown* contiene le seguenti routine:

QueryInterface	Fornisce i puntatori alle altre interfacce supportate dall'oggetto.
AddRef eRelease	Semplici metodi di conteggio dei riferimenti che tengono la traccia della durata dell'oggetto in modo che un oggetto possa cancellarsi quando il client non ha più bisogno del suo servizio.

Tramite il metodo *QueryInterface* di *IUnknown*, i client ottengono i puntatori ad altre interfacce. *QueryInterface* conosce le interfacce dell'oggetto server e può dare ad un client un puntatore all'interfaccia richiesta. Ricevendo un puntatore ad un'interfaccia, il client è certo di poter chiamare qualsiasi metodo dell'interfaccia.

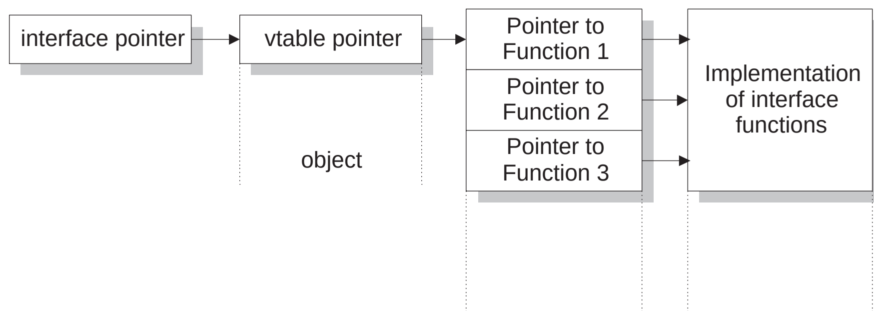
Gli oggetti registrano la propria durata tramite i metodi di *IUnknown*, *AddRef* e *Release*, che sono dei semplici metodi di conteggio di riferimento. Finché il conteggio di riferimento dell'oggetto è diverso da zero, l'oggetto rimane in memoria. Quando il conteggio di riferimento raggiunge lo zero, l'implementazione dell'interfaccia può disporre in modo sicuro degli oggetti sottostanti.

Puntatori di interfaccia COM

Un puntatore di interfaccia è un puntatore a un'istanza di un oggetto che punta, a sua volta, all'implementazione di ciascun metodo nell'interfaccia. L'implementazione è accessibile tramite un array di puntatori a questi metodi, chiamato **vtable**. Le vtable sono simili al meccanismo utilizzato per supportare le funzioni virtuali in C++. Grazie a questa somiglianza, il compilatore è in grado di risolvere le chiamate ai metodi nell'interfaccia con lo stesso criterio utilizzato per risolvere le chiamate ai metodi nelle classi C++.

La vtable viene condivisa tra tutte le istanze di una classe di oggetti, cosicché per ciascuna istanza dell'oggetto, il codice oggetto alloca una seconda struttura che contiene i suoi dati privati. Il puntatore di interfaccia del client, quindi, è un puntatore *al puntatore* a vtable, come mostrato nel seguente diagramma.

Figura 38.2 Vtable dell'interfaccia



In Windows 2000 e nelle successive versioni di Windows, quando un oggetto è in esecuzione in ambiente COM+, viene fornito un livello aggiunto di indirezione tra il puntatore all'interfaccia e il puntatore a vtable. Il puntatore all' interfaccia disponibile per il client punta a un interceptor, il quale a sua volta punta a vtable. Questo permette a COM+ di fornire servizi come l'attivazione just-in-time, per cui il server può essere disattivato e riattivato dinamicamente in un modo non visibile al client.Per ottenere ciò, COM+ garantisce che l'interceptor si comporti come se fosse un normale puntatore a vtable.

Server COM

Un server COM è un'applicazione o una libreria che fornisce servizi ad un'applicazione client o a una libreria. Un server COM è costituito da uno o più oggetti COM, dove un oggetto COM è un insieme di proprietà (membri dati o contenuto) e di metodi (o funzioni membro).

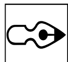
I client non sanno *come* un oggetto COM esegue il suo servizio; l'implementazione dell'oggetto rimane incapsulata. Un oggetto rende disponibili i suoi servizi tramite le sue **interfacce**, come descritto precedentemente.

Inoltre, i client non hanno bisogno di sapere *dove* risiede un oggetto COM. COM fornisce l'accesso trasparente che non tiene conto della **dislocazione** dell'oggetto.

Quando un client richiede un servizio da un oggetto COM, il client passa un identificatore di classe (CLSID) a COM. Un CLSID è semplicemente un GUID che identifica un oggetto COM. COM utilizza questo CLSID, memorizzato nel file di registro di sistema, per individuare l'apposita implementazione del server. Una volta individuato il server, COM porta il codice in memoria e fa in modo che il server crei un'istanza dell'oggetto per il client. Questo processo è gestito in modo indiretto tramite un oggetto speciale detto class factory o "fabbrica di classi" (basato sulle interfacce) che crea su richiesta istanze di oggetti.

In sostanza, un server COM deve:

- Registrare le voci nel registro di sistema che associano il modulo del server all'identificatore di classe (CLSID).
- Implementare un oggetto class factory, che costruisca un altro oggetto di un particolare CLSID.
- Esporre l'oggetto class factory a COM.
- Fornire un meccanismo di scaricamento tramite il quale un server che non sta fornendo servizi ai client possa essere rimosso dalla memoria.



I wizard di C++Builder automatizzano la creazione di oggetti e di server COM, come descritto nel paragrafo "Implementazione di oggetti COM con i wizard" a pagina 38-20.

CoClass e class factory

Un oggetto COM è un'istanza di una **CoClass**, che è una classe che implementa una o più interfacce COM. L'oggetto COM fornisce i servizi come definiti dalle sue interfacce.

Le CoClass vengono istanziate da uno speciale tipo di oggetto chiamato class factory. Ogni qualvolta i servizi di un oggetto vengono richiesti da un client, un class factory crea e registra un'istanza dell'oggetto per quel particolare client. Normalmente, se un altro client richiede i servizi dell'oggetto, il class factory crea un'altra istanza dell'oggetto per fornire il servizio al secondo client. (I client possono anche eseguire il bind ad oggetti COM in esecuzione, i quali si registrano al fine di poterli supportare.)

Una CoClass deve avere un class factory e un identificatore di classe (CLSID), in modo che possa essere istanziato esternamente, cioè, da un altro modulo. L'uso di questi particolari identificatori per le CoClass ne rende possibile l'aggiornamento ogni volta che le nuove interfacce vengono implementate nella loro classe. Una nuova interfaccia può modificare o aggiungere metodi senza influenzare le versioni più vecchie, il che è un problema comune quando si usano le DLL.

I wizard di C++Builder si occupano dell'assegnazione degli identificatori di classe, dell'implementazione e dell'istanziazione dei class factory.

Server in-process, out-of-process e remoti

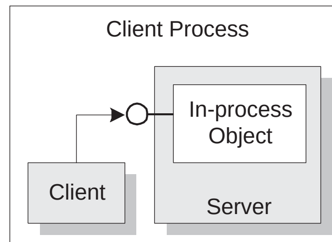
Con COM un client non ha bisogno di conoscere dove risiede un oggetto; semplicemente effettua una chiamata ad un'interfaccia dell'oggetto. COM compie i passi necessari per effettuare la chiamata. Questi sono diversi a seconda del fatto che

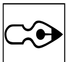
l'oggetto risieda nello stesso processo del client, in un altro processo nella macchina client o in un'altra macchina della rete. I diversi tipi di server sono conosciuti come:

Server in-process	Una libreria (DLL) che viene eseguita nello *stesso spazio di processo* del client, ad esempio, un controllo ActiveX incorporato in una pagina Web visualizzata in Internet Explorer o Netscape. Il controllo ActiveX viene, cioè, prelevato nella macchina client e richiamato all'interno dello stesso processo del browser Web. Il client comunica con il server in-process usando chiamate dirette all'interfaccia COM.
Server out-of-process (o server locale)	Un'altra applicazione (EXE) in esecuzione in un *differente spazio di processo*, ma sulla *stessa macchina* del client. Per esempio, un foglio elettronico Excel incorporato in un documento di Word sono due applicazioni separate in esecuzione sulla stessa macchina. Il server locale utilizza COM per comunicare con il client.
Server remoto	Una DLL o un'altra applicazione in esecuzione su una *macchina diversa* da quella del client. Per esempio, un'applicazione database C++Builder, collegata ad un application server su un'altra macchina della rete. Il server remoto usa interfacce COM (DCOM) distribuite per accedere e comunicare con l'application server.

Come mostrato nella Figura 38.3, per i server in-process i puntatori alle interfacce dell'oggetto sono nello stesso spazio di processo del client, quindi COM effettua chiamate dirette all'implementazione dell'oggetto.

Figura 38.3 Server in-process



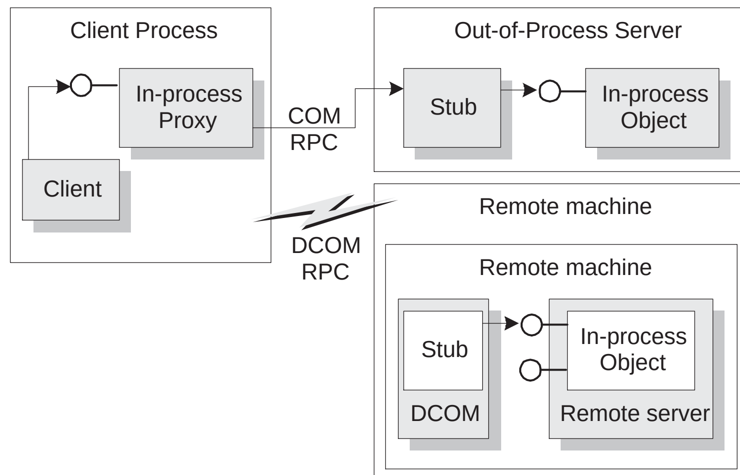


Ciò non è sempre vero in ambiente COM+. Quando un client effettua una chiamata a un oggetto in un contesto differente, COM+ intercetta la chiamata, facendo così in modo che essa si comporti come una chiamata a un server out-of-process (vedere più avanti) anche se il server è in-process. Per maggiori informazioni sulle operazioni con COM+, consultare il Capitolo 44, "Creazione di oggetti MTS o COM+".

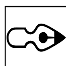
Come mostrato nella Figura 38.4, quando il processo si trova in un processo o in una macchina diversi, COM usa un proxy per iniziare le chiamate della procedura remota. Il **proxy** risiede nello stesso processo del client, quindi dal punto di vista del

client, tutte le chiamate all'interfaccia sembrano uguali. Il proxy intercetta la chiamata del client e la inoltra nel punto in cui l'oggetto vero e proprio è in esecuzione. Il meccanismo che consente al client di accedere agli oggetti in uno spazio di processo diverso, o anche in una macchina diversa, come se fossero nel proprio processo, è detto **marshaling** (smistamento**.**

Figura 38.4 Server out-of-process e remoti



La differenza tra i server out-of-process e quelli remoti sta nel tipo di comunicazione usata dai processi. Il proxy usa COM per comunicare con un server out-of-process e le interfacce COM (DCOM) distribuite per comunicare con una macchina remota. DCOM trasferisce, in modo del tutto trasparente, una richiesta di un oggetto locale ad un oggetto remoto in esecuzione su una macchina differente.



Per le chiamate a procedure remote, DCOM utilizza il protocollo RPC fornito da Open Group's Distributed Computing Environment (DCE). Per la sicurezza distribuita, DCOM utilizza il protocollo di sicurezza di NT LAN Manager (NTLM). Per i servizi delle directory, DCOM utilizza il Domain Name System (DNS).

Il meccanismo di smistamento

Lo smistamento è il meccanismo che consente ad un client di effettuare chiamate alle funzioni di interfaccia per oggetti remoti in un altro processo o su una macchina differente. Lo smistamento

- Prende un puntatore di interfaccia nel processo del server e rende disponibile un puntatore proxy al codice del processo del client.
- Trasferisce gli argomenti di una chiamata all'interfaccia come passati dal client e li colloca nello spazio di processo dell'oggetto remoto.

Per qualsiasi chiamata all'interfaccia, il client mette gli argomenti in uno stack ed effettua una chiamata di funzione tramite il puntatore di interfaccia. Se la chiamata all'oggetto non è in-process, questa viene passata al proxy. Il proxy chiude gli argomenti in un pacchetto di smistamento e trasmette la struttura all'oggetto remoto. Lo stub dell'oggetto apre il pacchetto, mette gli argomenti nello stack e chiama

l'implementazione dell'oggetto. In sintesi, l'oggetto ricrea la chiamata del client nel suo stesso spazio di indirizzo.

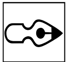
Il tipo di smistamento che si verifica dipende da quale interfaccia l'oggetto COM implementa. Gli oggetti possono usare un meccanismo di smistamento standard fornito dall'interfaccia *IDispatch*. Questo è un meccanismo di smistamento generico, che consente la comunicazione tramite una chiamata alla procedura remota tipica del sistema (RPC). Per informazioni dettagliate sull'interfaccia *IDispatch*, consultare il paragrafo "Interfacce di Automation" a pagina 41-13. Anche se l'oggetto non implementa *IDispatch*, se si limita a tipi compatibili con automation e possiede una libreria di tipi registrata, COM è in grado di fornire automaticamente il supporto per lo smistamento.

Le applicazioni che non si limitano ai tipi compatibili con automation o che non registrano una libreria di tipi devono provvedere al proprio smistamento. Lo smistamento viene fornito o attraverso un'implementazione dell'interfaccia *IMarshal*, o utilizzando una DLL proxy/stub generata separatamente. C++Builder non supporta la generazione automatica di DLL proxy/stub.

Aggregazione

A volte, un oggetto server utilizza un altro oggetto COM per compiere alcune funzioni. Per esempio, un oggetto gestione magazzino potrebbe fare uso di un oggetto fattura separato per gestire le fatture dei clienti. Se l'oggetto gestione magazzino desiderasse presentare l'interfaccia fattura ai client, ci sarebbe tuttavia un problema: Sebbene un client dotato dell'interfaccia magazzino possa chiamare *QueryInterface* per ottenere l'interfaccia fattura, al momento della creazione dell'oggetto fattura non sapeva niente dell'oggetto gestione magazzino e non può restituire un'interfaccia magazzino in risposta ad una chiamata a *QueryInterface*. Un client che ha l'interfaccia fattura non può rimettersi in contatto con l'interfaccia magazzino.

Per evitare questo problema, alcuni oggetti COM supportano l'**aggregazione**. Quando l'oggetto gestione magazzino crea un'istanza dell'oggetto fattura, gli passa una copia della propria interfaccia *IUnknown*. L'oggetto fattura può utilizzare quindi quella interfaccia *IUnknown* per gestire tutte le chiamate a *QueryInterface* che richiedono un'interfaccia, come l'interfaccia magazzino che egli non supporta. Quando accade ciò, i due oggetti insieme sono definiti un aggregato. L'oggetto fattura è detto oggetto interno, oppure oggetto contenuto dell'aggregato, e l'oggetto magazzino è detto oggetto esterno.



Per comportarsi come l'oggetto esterno di un aggregato, un oggetto COM deve creare l'oggetto interno utilizzando le funzioni *CoCreateInstance* o *CoCreateInstanceEx* delle API di Windows, passando il suo puntatore *IUnknown* come parametro che l'oggetto interno potrà utilizzare per chiamate *QueryInterface*. A questo scopo, è anche possibile creare un'istanza di un oggetto *TComInterface* e utilizzarne il metodo *CreateInstance*.

Gli oggetti che si comportano come oggetto interno di un aggregato mettono a disposizione due implementazioni dell'interfaccia *IUnknown*: una che rimanda le chiamate *QueryInterface* che non è in grado di gestire all'interfaccia di controllo *IUnknown* dell'oggetto esterno, e una che restituisce un errore quando riceve

chiamate *QueryInterface* che non è i grado di gestire. I wizard di C++Builder creano automaticamente oggetti che includono questo supporto per il comportamento da oggetti interni.

Client COM

I client sempre possono interrogare le interfacce di un oggetto COM per determinare cosa è i grado di fornire. Tutti gli oggetti COM consentono ai client di richiedere le interfacce note. Inoltre, se il server supporta l'interfaccia *IDispatch*, i client possono interrogare il server per ricevere informazioni sui metodi supportati dall'interfaccia. Gli oggetti server non possono fare previsioni sull'utilizzo dei propri oggetti da parte dei client. Analogamente, i client non hanno bisogno di sapere come (né tantomeno dove) un oggetto fornisce i servizi; semplicemente si affidano agli oggetti server per fornire i servizi che reclamizzano tramite le proprie interfacce.

Esistono due tipi di client COM, controller e contenitori. I controller lanciano il server e interagiscono con esso tramite la sua interfaccia. Richiedono servizi dall'oggetto COM oppure lo pilotano come processo separato. I contenitori ospitano i controlli visuali o gli oggetti che appaiono nell'interfaccia utente del contenitore. Utilizzano interfacce predefinite per trattare le questioni di visualizzazione con gli oggetti server. È impossibile avere una relazione di contenitori su DCOM; ad esempio, i controlli visuali che appaiono nell'interfaccia utente del contenitore devono essere situati localmente. Ciò è dovuto al fatto che ci si aspetta che i controlli si traccino da soli, il che richiede che abbiano accesso a risorse GDI locali.

C++Builder semplifica lo sviluppo di client COM consentendo l'importazione di una libreria di tipi o di un controllo ActiveX in un componente wrapper, in modo che l'oggetto server assomigli agli altri componenti della VCL. Per maggiori informazioni su questo processo, consultare Capitolo 40, "Creazione di client COM".

Estensioni COM

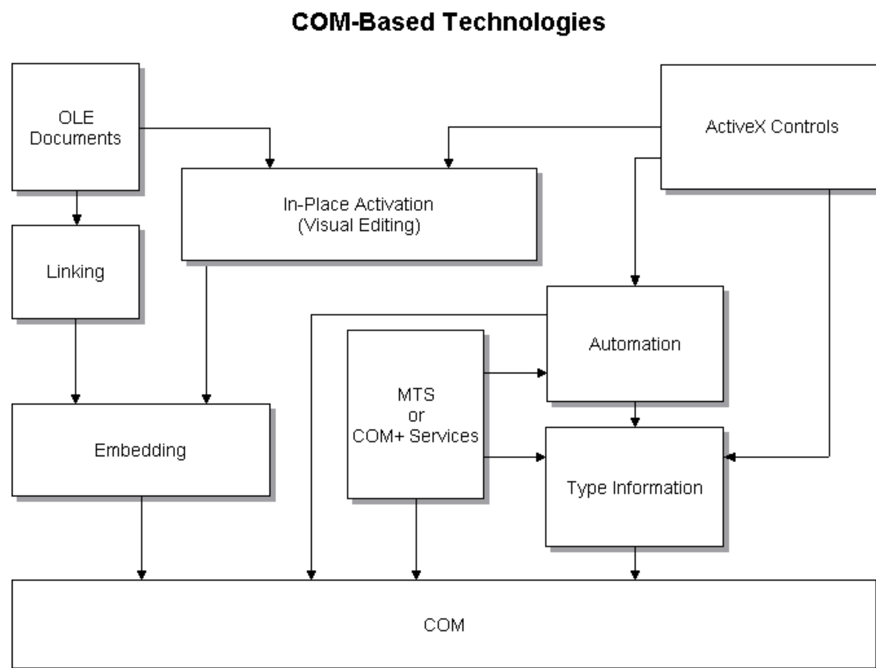
Originariamente COM è stato progettato per fornire una funzionalità centrale di comunicazione e per consentirne l'ampliamento tramite le estensioni. COM ha esteso poi la sua funzionalità centrale definendo set specialistici di interfacce per scopi specifici.

L'elenco seguente riporta alcune delle estensioni dei servizi COM forniti attualmente. Le sezioni successive descrivono questi servizi più in dettaglio.

Server Automation	Automation si riferisce alla possibilità di un'applicazione di controllare da programma gli oggetti di un'altra applicazione. Gli Automation server sono oggetti che possono essere controllati da altri eseguibili in fase di esecuzione.
Controlli ActiveX	I controlli ActiveX sono server in-process specializzati, progettati solitamente per essere incorporati in un'applicazione client. I controlli mettono a disposizione sia eventi sia comportamenti in esecuzione e in progettazione.
Active Server Pages	Active Server Pages sono script che generano pagine HTML. Questo linguaggio di scripting include costrutti per la creazione e l'esecuzione di oggetti Automation. Ciò significa che Active Server Page si comportano come un Automation controller.
Active Documents	Sono oggetti che supportano il collegamento e l'incorporamento, operazioni di drag-and-drop, l'editing visuale e l'attivazione in loco. I documenti di Word e i fogli elettronici di Excel sono esempi di Active Document.
Oggetti transazionali	Oggetti che includono un supporto supplementare per rispondere a un vasto numero di client. Questo supporto include funzioni quali attivazione just-in-time, transazioni, centralizzazione di risorse e servizi di sicurezza. In origine queste funzioni erano gestite da MTS ma, con l'avvento di COM+, sono state incorporate in COM.
Oggetti evento COM+ e subscription di evento	Oggetti che supportano il modello a eventi con associazione flessibile di COM+. A differenza del modello con associazione rigida usato dai controlli ActiveX, Il modello a eventi di COM+ consente di sviluppare publisher di eventi in modo indipendente dai subscriber di eventi.
Librerie di tipi	Una raccolta di strutture di dati statiche, spesso salvata come risorsa, che fornisce informazioni di tipo dettagliate su un oggetto e sulle sue interfacce. I client degli Automation server, dei controlli ActiveX e degli oggetti transazionali si aspettano che le informazioni di tipo siano disponibili.

Il seguente diagramma illustra la relazione delle estensioni COM e il modo in cui sono costruite in COM:

Figura 38.5 Tecnologie basate su COM



Gli oggetti COM possono essere visuali o non visuali. Alcuni devono essere eseguiti nello stesso spazio di processo dei rispettivi client; altri possono essere eseguiti in processi differenti o in macchine remote, purché gli oggetti forniscano il supporto per lo smistamento. La Tabella 38.1 riassume i tipi di oggetti COM che è possibile creare, indica se sono visuali o meno, gli spazi di processo in cui possono essere eseguiti, come provvedono allo smistamento e se richiedono o meno una libreria di tipi.

Tabella 38.1 Requisiti degli oggetti COM

Oggetto	È un oggetto visuale?	Spazio di processo	Comunicazione	Libreria di tipi
Active Document	Solitamente sì	Interno al processo o esterno al processo	Verbi OLE	No
Automation Server	A volte	In-process, out-of-process, oppure remoto	Smistamento automatico tramite l'interfaccia *IDispatch* (per i server esterni al processo e remoti)	Obbligatoria per lo smistamento automatico
ActiveX Control	Solitamente sì	Interno al processo	Smistamento automatico tramite l'interfaccia *IDispatch*	Obbligatoria

Tabella 38.1 Requisiti degli oggetti COM

Oggetto	È un oggetto visuale?	Spazio di processo	Comunicazione	Libreria di tipi
MTS o COM+	A volte	Interno al processo per MTS, qualsiasi per COM+	Smistamento automatico tramite libreria di tipi	Obbligatoria
Oggetti di interfaccia custom in-process	A volte	Interno al processo	Non è richiesto nessuno smistamento per i server interni al processo	Consigliata
Altri oggetti di interfaccia custom	A volte	In-process, out-of-process, oppure remoto	Smistamento automatico tramite una libreria di tipi; altrimenti smistamento manuale tramite interfacce personalizzate	Consigliata

Server Automation

Automation si riferisce alla possibilità di un'applicazione di controllare da programma gli oggetti di un'altra applicazione, come una macro che può manipolare più di un'applicazione contemporaneamente. L'oggetto server da manipolare viene chiamato oggetto Automation, mentre Il client di un oggetto Automation viene definito come Automation controller.

L'oggetto Automation può essere usato su server in-process, locali e remoti.

L'oggetto Automation è caratterizzato fondamentalmente da due aspetti:

- Definisce un set di proprietà e comandi e descrive le loro capacità attraverso le descrizioni di tipo. Per ottenere ciò, occorre trovare il modo di fornire informazioni sulle interfacce, sui metodi di interfaccia e sugli argomenti di questi metodi. Normalmente queste informazioni sono disponibili in una libreria di tipi. Automation server può generare anche informazioni di tipo dinamico, quando interrogato tramite la sua interfaccia *IDispatch* (vedere di seguito).
- Rende i metodi accessibili in modo che le altre applicazioni possano usarli. Per questo, implementa l'interfaccia *IDispatch*. Tramite questa interfaccia un oggetto può esporre tutti i suoi metodi e tutte le sue proprietà. Tramite il metodo principale di questa interfaccia è possibile richiamare i metodi dell'oggetto, dopo che sono stati identificati attraverso le informazioni di tipo.

Spesso gli sviluppatori utilizzano Automation per creare e utilizzare oggetti OLE non visuali che vengono eseguiti in tutti gli spazi di processo perché l'interfaccia *IDispatch* di Automation rende automatico il processo di smistamento. Automation, comunque, limita i tipi che è possibile utilizzare.

Per un elenco dei tipi che risultano validi per le librerie di tipi in generale, e per le interfacce Automation in particolare, consultare "Tipi consentiti" a pagina 39-12.

Per informazioni sulla scrittura di un Automation server, consultare il Capitolo 41, "Creazione di semplici server COM".

Active Server Pages

La tecnologia Active Server Page (ASP) consente di scrivere semplici programmi detti Active Server Pages che possono essere avviati dai client tramite un server web. A differenza dei controlli ActiveX che vengono eseguiti sui client, le Active Server Page sono eseguite sul server, e restituiscono come risultato ai client una pagina HTML.

Le Active Server Pages sono scritte in Jscript o in VB script. Lo script viene eseguito ogni volta che il server carica la pagina Web. Questi script possono avviare a loro volta un Automation server incorporato (o un Enterprise Java Bean). Per esempio, è possibile scrivere un Automation server, ad esempio per creare una bitmap o per eseguire una connessione a un database, e questo server accede ai dati che vengono aggiornati ogni qualvolta il client carica la pagina del Web.

Active Server Pages si basano sull'ambiente Microsoft Internet Information Server (IIS) per servire le pagine Web.

I wizard di C++Builder permettono di creare un Active Server Object, che è un oggetto Automation espressamente progettò per operare con un Active Server Page. Per maggiori informazioni sulla creazione e l'utilizzo di questi tipi di oggetti, consultare il Chapter 42, "Creazione di Active Server Page".

Controlli ActiveX

ActiveX è una tecnologia che consente di ottenere componenti COM, in particolar modo i controlli, più compatti ed efficienti. Questa caratteristica è particolarmente necessaria per quei controlli che si prevede dovranno essere utilizzati in applicazioni Intranet e che i client devono necessariamente scaricare prima di poterli utilizzare.

I controlli ActiveX sono controlli visuali che vengono eseguito solo come server in-process, e possono essere inglobati in un'applicazione contenitore di tipo controllo ActiveX. Essi non sono di per sé applicazioni complete, ma possono essere pensati come controlli OLE prefabbricati riutilizzabili in varie applicazioni. I controlli ActiveX hanno un'interfaccia utente visibile e devono appoggiarsi a interfacce predefinite per concordare le modalità di I/O e di visualizzazione con i contenitori che li ospitano.

I controlli ActiveX fanno uso di Automation per esporre le loro proprietà, i loro metodi e i loro eventi. Le funzionalità dei controlli ActiveX includono la possibilità di attivare gli eventi, di collegarsi alle sorgenti di dati e di supportare la concessione delle licenze.

Un uso dei controlli ActiveX è nei siti Web come oggetti interattivi di una pagina Web. Proprio per questo, ActiveX è diventato uno standard, destinato a contenuti interattivi per il World Wide Web, compreso l'uso degli ActiveX Document per la visualizzazione di documenti non HTML tramite un browser Web. Per ulteriori informazioni sulla tecnologia ActiveX, visitare il sito Web ActiveX di Microsoft.

I wizard di C++Builder consentono di creare con facilità controlli ActiveX. Per ulteriori informazioni sulla creazione e sull'uso di questi tipi di oggetti, consultare il Capitolo 43, "Creazione di un controllo ActiveX".

Active Documents

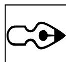
Gli Active Document (precedentemente definiti documenti OLE) costituiscono un set di servizi COM, che supportano il collegamento e l'incorporamento, il drag-and-drop, e l'editing visuale. Gli Active Document possono incorporare agevolmente dati o oggetti di formati differenti, quali clip sonore, spreadsheets, testo e bitmap.

A differenza dei controlli ActiveX, gli Active Document non sono limitati ai server in-process; possono essere usati anche in applicazioni cross-process.

A differenza degli oggetti Automation, che raramente sono visuali, gli oggetti Active Document possono essere attivi in modo visuale in un'altra applicazione. Pertanto, essi vengono associati a due tipi di dati: i dati di presentazione, usati per visualizzare l'oggetto su un monitor o su un dispositivo di output, e i dati nativi, usati per modificare un oggetto.

Gli oggetti Active Document possono essere contenitori o server di documenti. Anche se C++Builder non fornisce un wizard automatico per la creazione di Active Document, si può usare la classe VCL, *TOleContainer*, per supportare il collegamento e l'incorporamento degli Active Document esistenti.

È anche possibile usare *TOleContainer* come base per un contenitore Active Document. Per creare oggetti per i server Active Document, si deve usare il COM object wizard e aggiungere le interfacce appropriate, a seconda dei servizi che l'oggetto ha bisogno di supportare. Per ulteriori informazioni sulla creazione e sull'uso dei server Active Document, visitare il sito Web ActiveX di Microsoft.



Anche se la specifica di Active Document incorpora il supporto per lo smistamento relativamente alle applicazioni cross-process, gli Active Document non vengono eseguiti sui server remoti, poiché usano tipi specifici di un sistema su una determinata macchina come handle di finestre, handle di menu, e così via.

Oggetti transazionali

C++Builder utilizza il termine "oggetti transazionali" per fare riferimento a oggetti che sfruttano i servizi per le transazioni, la sicurezza e la gestione delle risorse forniti da Microsoft Transaction Server (MTS) (per le versioni di Windows precedenti a Windows 2000) o da COM+ (per Windows 2000 e successive). Questi oggetti sono progettati per operare in vasti ambienti distribuiti.

I servizi per le transazione forniscono la solidità, garantendo sempre il completamento delle attività o il ritorno alla stato di origine (il server non compie mai un'attività solo parzialmente). I servizi per la sicurezza consentono di esporre diversi livelli di supporto a classi diverse di client. La gestione delle risorse consente ad un oggetto di gestire più client unendo le risorse o mantenendo attivi gli oggetti soltanto quando sono utilizzati. Per consentire al sistema di fornire questi servizi, l'oggetto deve implementare l'interfaccia *IObjectControl*. Per accedere ai servizi, gli

oggetti transazionali usano un'interfaccia di nome *IObjectControl* creata per loro conto da MTS o da COM+.

In ambiente MTS, l'oggetto server deve essere incorporato in una libreria (DLL) che va quindi installata nell'ambiente di esecuzione di MTS. Ovvero, l'oggetto server è un server in-process che viene eseguito nel spazio del processo di esecuzione di MTS. In ambiente COM+, questa restrizione non esiste in quanto tutte le chiamate COM vengono indirizzate tramite un intercettatore. Dal punto di vista dei client, la differenza tra MTS e COM+ è inesistente.

I server MTS o COM+ raggruppano oggetti transazionali che vengono eseguiti nello stesso spazio di processo. In ambiente MTS, questo gruppo è definito un package MTS, mentre in ambiente COM+ è definito un'applicazione COM+. Una singola macchina può eseguire molti package MTS (o applicazioni COM+) diversi, dove ciascuno di essi viene eseguito in un spazio di processo separato.

Dal punto di vista dei client, l'oggetto transazionale può apparire come qualsiasi altro oggetto server COM. Il client non dovrà mai sapere nulla sulle transazioni, sulla sicurezza oppure sull'attivazione just-in-time, a meno che non stia iniziando una transazione per suo conto.

MTS e COM+ forniscono entrambi un strumento separato per amministrare gli oggetti transazionali. Questo strumento consente di configurare oggetti in package o applicazioni COM+, vedere package o applicazioni COM+ installate su un computer, vedere o modificare gli attributi degli oggetti inclusi, esaminare e gestire transazioni, rendere disponibili oggetti a client, e così via. In ambiente MTS, questo strumento è MTS Explorer. In ambiente COM+ è il COM+ Component Manager.

Oggetti evento COM+ e subscriber di eventi

Il sistema a eventi di COM+ introduce uno strato di software intermedio che separa le applicazioni che generano eventi (chiamate publisher) dalle applicazioni che rispondono agli eventi (dette subscriber). Invece di essere rigidamente collegati gli uni agli altri, i publisher e i subscriber possono essere sviluppati, distribuiti e attivati indipendentemente l'uno dall'altro.

Nel modello a eventi di COM+, viene creata per prima cosa un'interfaccia di evento utilizzando COM+ Event Object wizard di C++Builder's. L'interfaccia di evento non ha alcuna implementazione; semplicemente definisce i metodi dell'evento che i publisher genereranno e a cui i subscriber risponderanno. L'oggetto evento COM+ viene quindi installato in una COM+ Application, nel COM+ Catalog. L'operazione può essere condotta utilizzando l'IDE di C++Builder, da programma utilizzando l'oggetto *TComAdminCatalog* o da un un amministratore di sistema utilizzando lo strumento Component Services.

I subscriber di eventi sono responsabili della fornitura di un'implementazione per l'interfaccia dell'evento. È possibile creare componenti subscriber di eventi utilizzando COM+ Event Subscription wizard di C++Builder. Utilizzando il wizard, è possibile selezionare l'oggetto evento che si desidera implementare e C++Builder preparerà il modello di tutti i metodi dell'interfaccia. Nel caso l'oggetti evento non

sia stato ancora installato nel COM+ Catalog, è possibile anche selezionare una libreria di tipi.

Infine, una volta implementato il componente subscriber, lo si deve anche installare nel COM+ Catalog. Ancora una volta, l'operazione può essere condotta mediante l'IDE di C++Builder, mediante un oggetto *TComAdminCatalog* oppure utilizzando lo strumento amministrativo Component Services.

Quando un publisher desidera generare un evento, crea semplicemente un'istanza dell'oggetto evento (non il componente subscriber) e chiama i metodi appropriati dell'interfaccia dell'evento. A questo punto interviene COM+ e lo notifica a tutte le applicazioni che sono abbonate a quell'oggetto evento, chiamandole in modo sincrono una alla volta. In questo modo, i subscriber non devono avere alcuna informazione sulle applicazioni che si abbonano all'evento. Ai subscriber non serve nient'altro che un'implementazione dell'interfaccia dell'evento e non deveono far altro che selezionare quei publisher a cui desiderano abbonarsi. Tutto il resto viene gestito da COM+.

Per maggiori informazioni sulla generazione di eventi COM+, vedere "Generazione di eventi in ambiente COM+" a pagina 44-22.

Librerie di tipi

Le librerie di tipi forniscono un modo per ottenere ulteriori informazioni di tipo su un oggetto che può essere determinato dall'interfaccia di un oggetto. Le informazioni di tipo contenute nelle librerie di tipi forniscono i dati necessari sugli oggetti e sulle loro interfacce, comunicando, per esempio, quali interfacce esistono nei diversi oggetti (dato il CLSID), quali funzioni membro esistono in ciascuna interfaccia e quali argomenti richiedono tali funzioni.

È possibile ottenere informazioni di tipo sottoponendo a query un'istanza in esecuzione di un oggetto o caricando e leggendo le librerie di tipi. Con queste informazioni si può implementare un client per l'uso di un oggetto desiderato, sapendo specificamente quali funzioni membro occorrono, e che cosa passare loro.

I client degli Automation server, dei controlli ActiveX e degli oggetti transazionali si aspettano che le informazioni di tipo siano disponibili. Tutti wizard di C++Builder generano automaticamente una libreria di tipi. È possibile visualizzare o modificare queste informazioni di tipo usando il **Type Library Editor**, come descritto nel Capitolo 39, "Funzionamento delle librerie di tipi".

Questa sezione descrive che cosa contiene una libreria di tipi, come viene creata, quando viene usata e come vi si può accedere. Per gli sviluppatori che vogliono condividere le interfacce tra più linguaggi, la sezione termina con suggerimenti sull'uso degli strumenti della libreria di tipi.

Contenuto delle librerie di tipi

Le librerie di tipo contengono *informazioni di tipo*, che indicano quali interfacce esistono nei diversi oggetti COM, e i tipi e il numero degli argomenti per i metodi di interfaccia. Queste descrizioni includono identificatori univoci per le CoClass (CLSID) e per le interfacce (IID), in modo che vi si possa accedere in modo corretto,

nonché gli identificatori dispatch (dispID) per i metodi e le proprietà dell'interfaccia Automation.

Le librerie di tipi possono contenere anche le seguenti informazioni:

- Descrizione di informazioni personalizzate di tipo associate alle interfacce personalizzate
- Routine che vengono esportate dal server Automation o ActiveX, ma che non sono metodi di interfaccia
- Informazioni sulle enumerazioni, sui record (strutture), sulle unioni, sugli alias e sui tipi di dati del modulo
- Riferimenti alle descrizioni di tipo da altre librerie di tipo

Creazione di librerie di tipi

Con i tradizionali strumenti di sviluppo si possono creare librerie di tipi scrivendo script nell'Interface Definition Language (IDL) o nell'Object Description Language (ODL), quindi eseguendo lo script tramite un compilatore. Comunque, C++Builder genera automaticamente una libreria di tipi quando si crea un oggetto COM (inclusi controlli ActiveX, oggetti Automation, moduli dati remoti e così via) usando uno dei wizard presenti sulla pagina ActiveX o Multitier della finestra di dialogo New Items. È possibile creare una libreria di tipi anche scegliendo dal menu principale il comando File | New | Other, quindi selezionando la pagina ActiveX e quindi selezionando Type Library.

Per visualizzare la libreria di tipi, si può usare il Type Library editor di C++Builder. Una libreria di tipi può essere facilmente modificata usando il Type Library editor; C++Builder aggiorna automaticamente il file .tbl corrispondente (binary type library file) nel momento in cui si salva la libreria di tipi. Se si apportano modifiche alle Interfaces e alle CoClasses create utilizzando un wizard, il Type Library editor aggiorna anche i file di implementazione. Per maggiori informazioni sull'utilizzo del Type Library editor per scrivere interfacce e CoClasses, consultare il Capitolo 39, "Funzionamento delle librerie di tipi".

Quando usare le librerie di tipi

È importante creare una libreria di tipi per ogni insieme di oggetti che viene esposto agli utenti esterni,

- I controlli ActiveX richiedono una libreria di tipi, che deve essere inclusa come risorsa nella DLL che contiene i controlli ActiveX.
- Gli oggetti esposti che supportano il binding tramite vtable delle interfacce personalizzate devono essere descritti in una libreria di tipi, in quanto i riferimenti vtable vengono inglobati in fase di compilazione. I client importano le informazioni sulle interfacce dalla libreria di tipi e usano tali informazioni per la compilazione. Per ulteriori informazioni sul binding di vtable e su quello in fase di compilazione, consultare "Interfacce di Automation" a pagina 41-13.
- Le applicazioni che implementano gli Automation server devono fornire una libreria di tipi, in modo che i client vi si possano collegare anticipatamente.

- Gli oggetti istanziati dalle classi che supportano l'interfaccia *IProvideClassInfo*, devono avere una libreria di tipi.
- Le librerie di tipi non sono indispensabili, ma sono comunque utili per l'identificazione degli oggetti usati con OLE drag-and-drop.

Accesso alle librerie di tipi

La libreria binaria di tipi fa normalmente parte di un file di risorse (.res) o di un file indipendente con estensione .tlb. Se è inclusa in un file di risorse, la libreria di tipi può essere incorporata in un server (.dll, .ocx o .exe).

Una volta creata una libreria di tipi, i browser dell'oggetto, i compilatori e gli altri strumenti simili possono accedervi tramite speciali interfacce di tipo:

Interfaccia	*Descrizione*
ITypeLib	Fornisce i metodi per accedere alle descrizioni di una liberia di tipi.
ITypeLib2	Potenzia *ITypeLib* in modo da includere il supporto per stringhe di documentazione, dati custom e dati statistici sulla libreria di tipi.
ITypeInfo	Fornisce le descrizioni dei singoli oggetti contenuti in una libreria di tipi. Per esempio, un browser usa questa interfaccia per estrarre le informazioni sugli oggetti dalla libreria di tipi.
ITypeInfo2	Potenzia *ITypeInfo* in modo da accedere ad altre informazioni sulla libreria di tipi, inclusi i metodi per accedere ad elementi dati custom.
ITypeComp	Fornisce un modo rapido per accedere alle informazioni che servono al compilatore per il collegamento ad un'interfaccia.

C++Builder è in grado di importare e usare librerie di tipi da altre applicazioni scegliendo il comando Project|Import Type Library.

Vantaggi dell'uso delle librerie di tipi

Anche se un'applicazione non richiede una libreria di tipi, si possono sempre prendere in considerazione i vantaggi che derivano dal suo uso:

- È possibile usare il binding anticipato con Automation (anziché la chiamata tramite i Variant), mentre i controller che non supportano vtable o le interfacce duali possono inserire i dispID nel codice in fase di compilazione, migliorando le prestazioni in fase di esecuzione.
- I browser di tipo possono analizzare la libreria, in modo che i client possano vedere le caratteristiche degli oggetti.
- Si può usare la funzione *RegisterTypeLib* per registrare gli oggetti esposti nel database di registrazione.
- Si può usare la funzione *UnRegisterTypeLib* per disinstallare completamente una libreria di tipi dell'applicazione dal registro di sistema.

- L'accesso al server locale viene migliorato, in quanto Automation usa le informazioni della libreria di tipi per inglobare i parametri che vengono passati ad un oggetto in un altro processo.

Uso degli strumenti della libreria di tipi

Qui di seguito sono elencati gli strumenti per lavorare con le librerie di tipi.

- Lo strumento TLIBIMP (Type Library Import), che prende le librerie di tipi esistenti e crea i file C++Builder Interface (file _TLB.cpp e _TLB.h), è incluso nel Type Library editor. TLIBIMP offre altre opzioni di configurazione che non sono disponibili nel Type Library editor.
- TRegSvr è uno strumento per la registrazione e la rimozione dei server e delle librerie di tipi, che viene fornito con C++Builder. Il file sorgente di TRegSvr è disponibile come esempio nella directory Examples.
- Il compilatore Microsoft IDL (MIDL) compila gli script IDL per creare una libreria di tipi. MIDL ha un'opzione facoltativa per la generazione di file header, trovata in MS Win32 SDK.
- RegSvr32.exe, uno strumento per la registrazione e la rimozione dei server e delle librerie di tipi, è un programma di utilità standard di Windows.
- OLEView OLEView è uno strumento browser della libreria di tipi, disponibile nel sito Web di Microsoft.

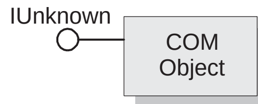
Implementazione di oggetti COM con i wizard

C++Builder semplifica enormemente la scrittura di applicazioni server COM poiché include diversi wizard in grado di gestire la maggior parte dei dettagli. C++Builder mette a disposizione wizard distinti per creare:

- Un semplice oggetto COM
- Un oggetto Automation
- Un Active Server Object (per l'incorporazione in una Active Server page)
- Un controllo ActiveX
- Una scheda ActiveX
- Un oggetto transazionale
- Un oggetto COM+ Event
- Un oggetto COM+ Subscription Event
- Una pagina di proprietà
- Una libreria di tipi
- Una libreria ActiveX

I wizard gestiscono la maggior parte delle operazioni connesse alla creazione di qualunque tipo di oggetto COM. Forniscono le interfacce COM necessarie per ciascun tipo di oggetto. Come mostrato nella Figura 38.6, con un semplice oggetto COM il wizard implementa l'interfaccia COM *IUnknown* necessaria, che fornisce un puntatore di interfaccia all'oggetto.

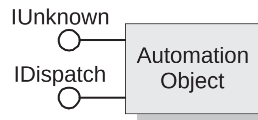
Figura 38.6 Una semplice interfaccia per un oggetto COM



Il COM object wizard fornisce anche una implementazione per *IDispatch* nel caso si specifichi che si sta creando un oggetto che supporta un discendente di *IDispatch*.

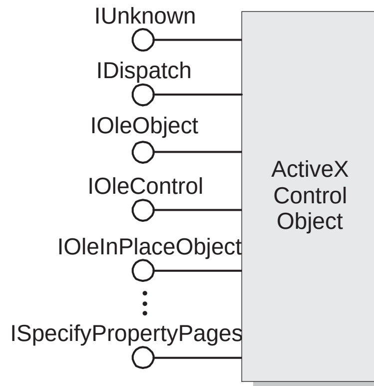
Come mostrato nella Figura 38.7, per gli oggetti Automation e Active Server, il wizard implementa *IUnknown* e *IDispatch*, che forniscono lo smistamento automatico.

Figura 38.7 Interfaccia di un oggetto Automation



ACome mostrato nella Figura 38.8, per gli oggetti del controllo ActiveX e schede ActiveX il wizard implementa tutte le necessarie interfacce del controllo ActiveX, *IUnknown*, *IDispatch*, *IOleObject*, *IOleControl*, e così via.

Figura 38.8 Interfaccia di un oggetto ActiveX



La Tabella 38.2 elenca i vari wizard e le interfacce da essi implementate:

Tabella 38.2 Wizard di C++Builder per l'implementazione di oggetti COM, Automation e ActiveX

Wizard	Interfacce implementate	Che cosa fa il wizard
COM server	IUnknown (e *IDispatch* se si seleziona un'interfaccia predefinita che deriva da *IDispatch*)	Esporta le routine per gestire la registrazione del server, la registrazione della classe, il caricamento e lo scaricamento del server, e l'istanziazione di un oggetto. Crea e gestisce le class factory per gli oggetti implementati nel server. Prepara gli elementi del file di registro per quegli oggetti che specificano il modello di threading selezionato. Offre il supporto lato-server per la generazione di eventi, se necessario. Dichiara i metodi che implementano una interfaccia selezionata, e preparando la struttura base delle implementazioni che il programmatore dovrà poi completare. Fornisce una libreria di tipi. Permette di selezionare un'interfaccia arbitraria, registrata nella libreria di tipi, e di implementarla. Se lo si fa, è necessario utilizzare una libreria di tipi.
Automation server	*IUnknown*, *IDispatch*	Esegue le funzioni di un wizard per server COM (descritto in precedenza), e inoltre: Implementa l'interfaccia specificata, duale o dispatch.
Active Server Object	*IUnknown*, *IDispatch*, *(IASPObject)*	Esegue le funzioni di un wizard per oggetti Automation (descritto in precedenza) e genera facoltativamente una pagina .ASP che può essere caricata in un browser Web. Lascia aperto il Type Library editor per consentire, se occorre, la modifica delle proprietà dell'oggetto e dei metodi. Mette a disposizione le caratteristiche intrinseche di ASP sotto forma di proprietà in modo da semplificare il reperimento delle informazioni relative all'applicazione ASP e ai messaggi HTTP che l'hanno avviata.
ActiveX Control	*IUnknown*, *IDispatch*, *IPersistStreamInit*, *IOleInPlaceActiveObject*, *IPersistStorage*, *IViewObject*, *IOleObject*, *IViewObject2*, *IOleControl*, *IPerPropertyBrowsing*, *IOleInPlaceObject*, *ISpecifyPropertyPages*	Esegue le funzioni del wizard per Automation server (descritti in precedenza), e inoltre: Genera una CoClass di implementazione che corrisponde al controllo della VCL su cui è basato il controllo ActiveX e che implementa tutte le interfacce ActiveX. Lascia aperto l'editor del codice sorgente in modo che si possa modificare la classe di implementazione.

Tabella 38.2 Wizard di C++Builder per l'implementazione di oggetti COM, Automation e ActiveX (continua)

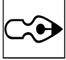
Wizard	Interfacce implementate	Che cosa fa il wizard
ActiveForm	Stesse interfacce di ActiveX Control	Esegue le funzioni del wizard per controlli ActiveX, e inoltre: Crea un discendente di *TActiveForm* che sostituisce la classe VCL preesistente nel wizard ActiveX control. Questa nuova classe consente di progettare la Active Form con le stesse modalità usate per progettare una scheda in una applicazione Windows.
Transactional object	*IUnknown*, *IDispatch*, *IObjectControl*	Aggiunge al progetto corrente una nuova unit che contiene la definizione dell'oggetto MTS o COM+. Inserisce nella libreria di tipi GUID proprietari in modo che C++Builder possa installare correttamente l'oggetto, lasciando aperto il Type Library editor per consentire la definizione delle interfacce che l'oggetto espone ai client. L'oggetto deve essere installato separatamente dopo averlo generato.
Property Page	*IUnknown*, *IPropertyPage*	Crea una nuova pagina di proprietà che è possibile progettare nel Forms designer.
COM+ Event object	Nessuna, per impostazione predefinita	Crea un oggetto evento COM+ che è possibile definire usando il Type Library editor. A differenza dei wizard per gli altri oggetti, COM+ Event object wizard non crea una unit di implementazione in quanto gli oggetti evento non hanno implementazione (che è fornita dall'oggetto subscriber dell'evento).
COM+ Subscription object	Nessuna, per impostazione predefinitadefault	Crea un oggetto subscriber COM+ che implementa l'interfaccia di evento scelta.
Type Library	Nessuna, per impostazione predefinita	Crea una nuova libreria di tipi e la associa al progetto attivo.
ActiveX library	Nessuna, per impostazione predefinita	Crea un nuovo ActiveX o server Com DLL ed espone le necessarie funzioni di esportazione.

È possibile aggiungere altri oggetti COM o reimplementare una loro implementazione esistente. Per aggiungere un nuovo oggetto, la cosa più semplice è utilizzare nuovamente un wizard. Questo perché il wizard imposta un'associazione fra la libreria di tipi e una classe di implementazione, in modo che le modifiche apportate alla libreria di tipi vengano applicate automaticamente all'oggetto di implementazione.

Codice generato dai wizard

I wizard di C++Builder generano codice che utilizza il linguaggio ATL (Active Template Library) di Microsoft come base per il supporto COM. ATL è un framework di classi modello che gestiscono parecchi dettagli implementativi dello sviluppo di applicazioni COM.Poiché ATL si basa su modelli, non è possibile eseguire il link in DLL. Il progetto includerà, invece, i file header di ATL che saranno

compilati in codice oggetto. I wizard di C++Builder generano le istruzioni include per questi file header in una unit con suffisso _ATL (Per esempio, Project1_ATL.cpp e Project1_ATL.h).



I file header ATL presenti nella directory Include/ATL di C++Builder differiscono leggermente dai file header di ATL forniti da Microsoft. Queste differenze sono necessarie affinché il compilatore di C++Builder possa compilare gli header. Non è possibile sostituire questi header con un'altra versione di ATL, pena la non corretta compilazione.

Oltre alle istruzioni include per i file ATL (e per i file aggiuntivi che consentono alle classi ATL di operare con le classi VCL), lo header della unit _ATL generato include anche la dichiarazione di una variabile globale di nome _Module. _Module è un'istanza della classe ATL *CComModule* che isola il resto dell'applicazione dalle differenze tra DLL e EXE nel modo in cui sono gestiti gli aspetti relativi al threading e alla registrazione. Nel file di progetto (Project1.cpp), _Module viene assegnata a un'istanza di *TComModule*, un discendente di *CComModule* che supporta lo stile di C++Builder per la registrazione di COM. Di solito, non sarà necessario utilizzare direttamente questo oggetto.

Il wizard aggiunge al file di progetto anche una mappa di oggetti. Questa è una serie di macro ATL che assomigliano a quanto segue:

```
BEGIN_OBJECT_MAP(ObjectMap)
  OBJECT_ENTRY(CLSID_MyObj, TMyObjImpl)
END_OBJECT_MAP()
```

Ogni elemento tra le linee BEGIN_OBJECT_MAP e END_OBJECT_MAP definisce un'associazione tra un ID di classe e la sua classe di implementazione ATL. L'oggetto _Module usa questa mappa per registrare i componenti. La mappa di oggetti è utilizzata anche dalle classi creatore di oggetti ATL. Se si aggiunge all'applicazione un qualsiasi oggetto COM senza l'aiuto di un wizard, si dovrà aggiornare la mappa degli oggetto affinché possano essere registrati e creati nel corretto. Per fare ciò, aggiungere un'altra linea che utilizzi la macro OBJECT_ENTRY, assegnando come parametri l'ID di classe e il nome della classe di implementazione.

I wizard generano una unit di implementazione per il particolare tipo di oggetto COM che si sta creando. L'unit di implementazione contiene la dichiarazione della classe che implementa l'oggetto COM. Questa classe è un discendente (diretto o indiretto) della classe ATL *CComObjectRootEx*, della classe ATL *CComCoClass*, e di altre classi che dipendono dal tipo di oggetto che si sta creando.

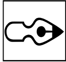
CComCoClass fornisce il supporto class factory necessario per la creazione della classe. Utilizza la mappa di oggetti che era stata aggiunta al file di progetto.

CComObjectRootEx è una classe modello che fornisce il supporto necessario per *IUnknown*. Essa implementa il metodo *QueryInterface* facendo uso di una mappa di interfacce che è possibile trovare nell'header della unit di implementazione. La mappa di interfacce assomiglia a quanto segue:

```
BEGIN_COM_MAP(TMyObjImpl)
  COM_INTERFACE_ENTRY(IMyObj)
END_COM_MAP()
```

Ogni elemento nella mappa di interfacce è un'interfaccia che viene esposta dal metodo *QueryInterface*. Se si aggiungono alla classe di implementazione ulteriori interfacce, sarà necessario aggiungerle alla mappa di interfacce (utilizzando la macro COM_INTERFACE_ENTRY), e aggiungerle come altri antenati della classe di implementazione.

CComObjectRootEx fornisce il supporto necessario per il conteggio dei riferimenti. Tuttavia, non dichiara i metodi *AddRef* e *Release*. Questi metodi vengono aggiunti alla classe di implementazione tramite la macro END_COM_MAP() alla fine della mappa di interfacce.



Per maggiori informazioni su ATL, consultare la documentazione di Microsoft. Si noti, comunque, che il supporto COM di C++Builder non utilizza ATL per la registrazione, per i controlli ActiveX (che sono invece basati su oggetti della VCL), o per il supporto di pagina di proprietà.

I wizard generano anche una libreria di tipi e la relativa unit associata, il cui nome assume il formato Project1_TLB.La unit Project1_TLB include le definizioni che l'applicazione ha bisogno di utilizzare, le definizioni di tipo e le interfacce definite nella libreria di tipi. Per maggiori informazioni sui contenuti di questo file, consultare "Codice generato durante l'importazione delle informazioni di una libreria di tipi" a pagina 40-5.

È possibile modificare l'interfaccia generata dal wizard utilizzando il Type Library editor. Se si apportano modifiche, la classe di implementazione viene automaticamente aggiornata in modo da tener conto di tali modifiche. Per completare l'implementazione, sarà necessario solamente completare il corpo dei metodi generati.

Capitolo

39

Funzionamento delle librerie di tipi

Questo capitolo descrive come creare e modificare le librerie di tipi usando l'editor di Type Library di C++Builder. Le librerie di tipi sono file che includono informazioni sui tipi di dati, sulle interfacce, sulle funzioni membro e sulle classi di oggetti esposte da un oggetto COM. Le librerie di tipi forniscono un modo per identificare quali tipi di oggetti e di interfacce sono disponibili su un server. Per una panoramica dettagliata sul perché e su quando usare le librerie di tipi, consultare "Librerie di tipi" a pagina 38-17.

Una libreria di tipi può contenere:

- Informazioni sui tipi di dati personalizzati come gli alias, gli enumerativi, le strutture e le union.
- Descrizioni su uno o più elementi COM, come un'interfaccia, dispinterface, o una CoClass. A ciascuna di queste descrizioni viene generalmente fatto riferimento come *informazioni di type.*
- Descrizioni delle costanti e dei metodi definiti in moduli esterni.
- Riferimenti alle descrizioni di tipo di altre librerie di tipi.

Includendo una libreria di tipi con l'applicazione COM o con la libreria ActiveX, si rendono disponibili le informazioni sugli oggetti dell'applicazione ad altre applicazioni e ad altri strumenti di programmazione tramite gli strumenti per le librerie di tipi e le interfacce di COM.

Con i tradizionali strumenti di sviluppo, è possibile preparare librerie di tipi creando script in IDL (Interface Definition Language), quindi eseguendoli tramite un compilatore. Il Type Library editor automatizza alcuni di questi processi, rendendo molto meno complessa ed onerosa la creazione di proprie librerie di tipi.

Quando si crea un server COM di qualsiasi tipo (controllo ActiveX, oggetto Automation, modulo dati remoto e così via) utilizzando i wizard di C++Builder, il wizard genera automaticamente una libreria di tipi. La maggior parte del lavoro che si fa per personalizzare l'oggetto generato inizia con la libreria di tipi, poiché è in essa che si definiscono le proprietà e i metodi che la libreria espone ai client: si modifica

l'interfaccia della CoClass generata dal wizard utilizzando il Type Library editor. Il Type Library editor aggiorna automaticamente la unit di implementazione dell'oggetto, e tutto ciò che si dovrà fare sarà completare il corp dei metodi generati.

Type Library editor

Il Type Library editor è uno strumento che consente agli sviluppatori di esaminare e di creare informazioni di tipo per gli oggetti COM. L'utilizzo del Type Library editor può semplificare enormemente lo sviluppo di oggetti COM centralizzando la definizione di interfacce, CoClasses, e tipi, l'ottenimento dei GUID per le nuove interfacce, l'associazione di interfacce alle CoClasses, l'aggiornamento della unit di implementazione e così via.

Il Type Library editor genera due tipi di file che rappresentano i contenuti della libreria di tipi:

Tabella 39.1 File di Type Library editor

File	Descrizione
.TLB file	Il file binario della libreria di tipi. Per impostazione predefinita, non sarà necessario utilizzare questo file, poiché la libreria di tipi viene compilata automaticamente nell'applicazione come risorsa. Comunque, sarà possibile utilizzare questo file per compilare esplicitamente la libreria di tipi in un altro progetto o per distribuire separatamente la libreria di tipi dall'EXE o dall'OCX. Per maggiori informazioni, consultare "Apertura di una libreria di tipi esistente" a pagina 39-14 and "Distribuzione delle librerie di tipi" a pagina 39-20.
_TLB unit	Questa unit (file .cpp e .h) interpreta i contenuti della libreria di tipi per l'uso nell'applicazione. Contiene tutte le dichiarazioni che occorrono all'applicazione per utilizzare gli elementi definiti nella libreria di tipi. Sebbene sia possibile aprire questo file nel Code editor, non lo si dovrà mai modificare -- la manutenzione è affidata al Type Library editor, e pertanto tutte le modifiche apportate saranno sovrascritte dal Type Library editor. Per maggiori informazioni sui contenuti di questo file, consultare "Codice generato durante l'importazione delle informazioni di una libreria di tipi" a pagina 40-5.

Elementi del Type Library editor

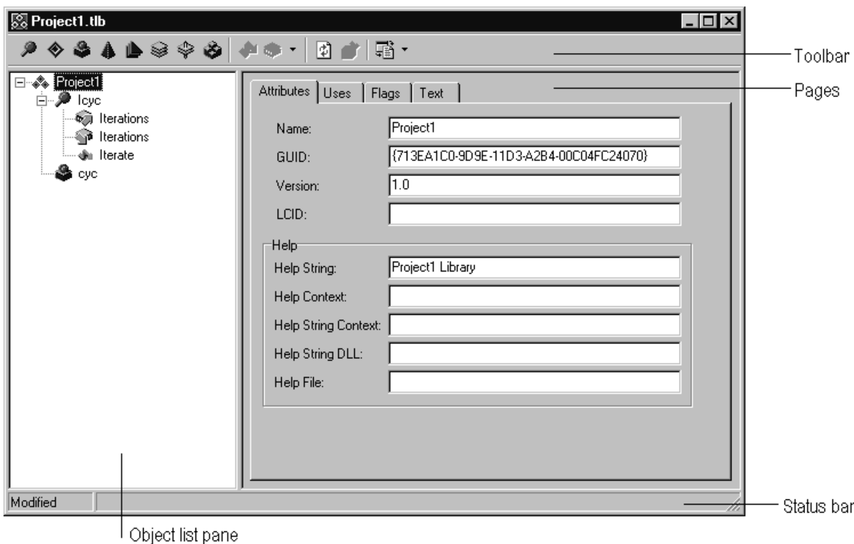
Gli elementi principali del Type Library editor sono elencati nella Tabella 39.2:

Tabella 39.2 Parti di Type Library editor

Parte	Descrizione
Barra strumenti	Include pulsanti per aggiungere alla libreria di tipi nuovi tipi, CoClass, interfacce e membri dell'interfaccia. La barra strumenti include anche pulsanti per aggiornare la unit di implementazione, registrare la libreria di tipi, e salvare un file IDL contenente le informazioni incluse nella libreria di tipi.
Pannello Object list	Mostra tutti gli elementi esistenti nella libreria di tipi. Quando si fa clic su uno degli elementi nel pannello, vengono visualizzate la pagine valide relative a quell'oggetto.
Barra di stato	Visualizza gli errori di sintassi se si cerca di aggiungere tipi non validi alla libreria di tipi.
Pagine	Visualizzano informazioni sull'oggetto selezionato. Le pagine visualizzate dipendono dal tipo di oggetto selezionato.

Tali parti sono illustrate nella Figura 39.1, che mostra il Type Library editor che visualizza le informazioni di tipo per un oggetto COM di nome cyc.

Figura 39.1 Type Library editor



Barra strumenti

La barra strumenti del Type Library editor, posizionata nella parte superiore del Type Library editor, contiene i pulsanti che è possibile selezionare per aggiungere alla libreria di tipi nuovi oggetti.

Il primo gruppo di pulsanti consente di aggiungere elementi alla libreria di tipi. Se si fa clic su un pulsante della barra strumenti, nel pannello Object list appaiono le icone

relative a quel certo elemento. È possibile personalizzarne gli attributi utilizzando il pannello di destra. A seconda dell'icona selezionata, sulla destra appaiono pagine di informazioni differenti.

La tabella seguente elenca gli elementi che è possibile aggiungere alla libreria di tipi:

Icona	Significato
	Una descrizione di interfaccia.
	Una descrizione di dispinterface.
	Una CoClass.
	Una enumerazione.
	Un alias.
	Un record.
	Una union.
	Un modulo.

Quando si seleziona nel pannello Object list uno degli elementi elencati in precedenza, il secondo gruppo di pulsanti visualizza i membri che risultano validi per quel certo elemento. Per esempio, quando si seleziona Interface, le icone Method e Property nel secondo riquadro diventano attive poiché è possibile aggiungere alla definizione di interfaccia metodi e proprietà. Quando si seleziona Enum, il secondo gruppo di pulsanti cambia per visualizzare il membro Const, che è l'unico membro valido per le informazioni di tipo Enum.

La tabella seguente elenca i membri che è possibile aggiungere agli elementi nel pannello Object list

Icona	Significato
	Un metodo dell'interfaccia, della dispinterface o un entry point in un modulo.
	Una proprietà di un'interfaccia o di una dispinterface.
	Una proprietà a sola scrittura. (disponibile dall'elenco a discesa sul pulsante della proprietà)
	Una proprietà in lettura/scrittura. (disponibile dall'elenco a discesa sul pulsante della proprietà)
	Una proprietà a sola lettura. (disponibile dall'elenco a discesa sul pulsante della proprietà)

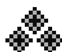
 Un campo in un record o in una union.

 Una costante in una enumerazione o in un modulo.

Nel terzo riquadro è possibile aggiornare, registrare o esportare la propria libreria di tipi (salvandola come file IDL) come descritto nel paragrafo “Salvataggio e registrazione delle informazioni della libreria di tipi” a pagina 39-19.

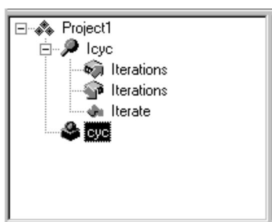
Pannello Object list

Il pannello Object list mostra tutti gli elementi della libreria di tipi attiva utilizzando una vista ad albero. La radice dell’albero rappresenta la libreria ed è rappresentata con la seguente icona:



Dal nodo della libreria di tipi si diramano verso il basso gli elementi inclusi nella libreria di tipi:

Figura 39.2 Pannello Object list



Quando si seleziona uno di questi elementi (incluso la stessa libreria di tipi), le pagine relative alle informazioni di tipo riportate alla destra si modificano riportando solo le informazioni pertinenti per quel certo elemento. È possibile utilizzare queste pagine per modificare la definizione e le proprietà dell’elemento scelto.

È possibile intervenire sugli elementi nel pannello Object list facendo clic destro in modo da richiamare il menu contestuale. In questo menu sono inclusi comandi che permettono di utilizzare gli Appunti di Windows per spostare o copiare gli elementi esistenti, oltre a comandi per aggiungere nuovi elementi o per personalizzare l’aspetto del Type Library editor.

Barra di stato

Quando si effettua l’editing o il salvataggio di una libreria di tipi, la sintassi, gli errori di traduzione e i messaggi di segnalazione vengono elencati nel pannello Status.

Per esempio, se si specifica un tipo IDL non supportato dal Type Library editor, viene visualizzato un errore di sintassi. Per un elenco completo dei tipi supportati dal type Library editor, consultare “Tipi consentiti” a pagina 39-12.

Pagine di informazioni di tipo

Quando si seleziona un elemento nel pannello Object list, nel Type Library editor appaiono le pagine di informazioni di tipo che sono valide per l'elemento selezionato. Quali pagine appariranno dipende dall'elemento selezionato nel pannello Object list:

Tabella 39.3 Pagine del Type Library editor

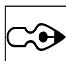
Elemento Type Info	Pagina di informazione di tipo	Contenuto della pagina
Type library	Attributes	Nome, versione e GUID della libreria di tipi, oltre a informazioni per il collegamento alla Guida.
	Uses	Elenco di altre librerie di tipi che contengono definizioni da cui questa libreria dipende.
	Flags	Flag che determinano come le altre applicazioni possono utilizzare la libreria di tipi.
	Text	Tutte le definizioni e le dichiarazioni che definiscono la libreria di tipi in oggetto (vedere di seguito).
Interface	Attributes	Denominate, versione e GUID dell'interfaccia, il nome dell'interfaccia da cui deriva, e informazioni utili per il link dell'interfaccia.
	Flags	Flag che indicano se l'interfaccia è nascosta, duale, compatibile con Automation e/o estensibile.
	Text	Le definizioni e le dichiarazioni per l'Interface (trattatta nel prosieguo del capitolo).
Dispinterface	Attributes	Nome, versione e GUID dell'interfaccia, e informazioni per il collegamento alla Guida.
	Flags	Flag che indicano se la Dispinterface è nascosta, duale, e/o estensibile.
	Text	Le definizioni e le dichiarazioni della Dispinterface. (vedere di seguito).
CoClass	Attributes	Nome, versione e GUID della CoClass e informazioni per il collegamento alla Guida.
	Implements	Un elenco di interfacce implementate dalla CoClass e dei rispettivi attributi.
	COM+	Gli attributi di oggetti transazionali, come il modello della transazione, la sincronizzazione delle chiamate, l'attivazione just-in-time, la centralizzazione di oggetti e così via. Include anche gli attributi degli oggetti evento di COM+.
	Flags	Flag che indicano i vari attributi della CoClass, incluso come i client possono creare e utilizzare istanze, se è visualizzabile dagli utenti in un browser, se è un controllo ActiveX, e se può essere aggregato (agisce come parte di un composito).
	Text	Le definizioni e le dichiarazioni per la CoClass (trattatta nel prosieguo del capitolo).
Enumeration	Attributes	Nome, versione e GUID della enumerazione, e informazioni per il collegamento alla Guida.
	Text	Le definizioni e le dichiarazioni per il tipo enumerativo (trattatto nel prosieguo del capitolo).

Tabella 39.3 Pagine del Type Library editor

Elemento Type Info	Pagina di informazione di tipo	Contenuto della pagina
Alias	Attributes	Nome, versione e GUID della enumerazione, il tipo che l'alias rappresenta e informazioni per il collegamento alla Guida.
	Text	Le definizioni e le dichiarazioni per l'alias (trattatto nel prosieguo del capitolo).
Record	Attributes	Nome, versione e GUID del record e informazioni per il collegamento alla Guida.
	Text	Le definizioni e le dichiarazioni per il record (trattatto nel prosieguo del capitolo).
Union	Attributes	Nome, versione e GUID della union e informazioni per il collegamento alla Guida.
	Text	Le definizioni e le dichiarazioni per la union (trattatta nel prosieguo del capitolo).
Module	Attributes	Nome, versione, GUID e DLL associata per il modulo e informazioni per il collegamento alla Guida.
	Text	Le definizioni e le dichiarazioni per il modulo (trattatto nel prosieguo del capitolo).
Method	Attributes	Nome, ID di smistamento o punto d'ingresso della DLL e informazioni per il collegamento alla Guida.
	Parameters	Tipo restituito del metodo, e un elenco di tutti i parametri con relativi tipi ed eventuali modificatori.
	Flags	Flag che indicano come i client possono vedere e utilizzare il metodo, se questo è un metodo predefinito per l'interfaccia e se è ricollocabile.
	Text	Le definizioni e le dichiarazioni per il metodo (trattatto nel prosieguo del capitolo).
Property	Attributes	Nome, ID di smistamento, tipi di metodo di accesso alla proprietà (getter o setter) e informazioni per il collegamento alla Guida.
	Parameters	Tipo restituito del metodo di accesso alla proprietà e un elenco di tutti i parametri con relativi tipi ed eventuali modificatori.
	Flags	Flag che indicano come i client possono vedere e utilizzare la proprietà, se questa è un'impostazione predefinita per l'interfaccia, se la proprietà è ricollocabile, collegabile, e così via.
	Text	Le definizioni e le dichiarazioni per il metodo di accesso alle proprietà (trattatto nel prosieguo del capitolo).
Const	Attributes	Nome, valore, tipo (per const di modulo) e informazioni per il collegamento alla Guida.
	Flags	Flag che indicano come i client possono vedere e utilizzare la costante, se questa rappresenta un valore predefinito, se la costante è collegabile, e così via.
	Text	Le definizioni e le dichiarazioni per la costante (trattatta nel prosieguo del capitolo).
Field	Attributes	Nome, tipo e informazioni per il collegamento alla Guida.

Tabella 39.3 Pagine del Type Library editor

Elemento Type Info	Pagina di informazione di tipo	Contenuto della pagina
	Flags	Flag che indicano come i client possono vedere e utilizzare il campo, se questo rappresenta un valore predefinito, se il campo è collegabile, e così via.
	Text	Le definizioni e le dichiarazioni per il campo (trattatto nel prosieguo del capitolo).



Per informazioni più particolareggiate sulle varie opzioni che è possibile impostare nelle pagine delle informazioni di tipo, consultare la Guida in linea del Type Library editor.

È possibile utilizzare tutte le pagine delle informazioni di tipo per vedere o modificare i valori in esse visualizzate.La maggior parte delle pagine organizza le informazioni in una serie di controlli, consentendo sia l'immissione di valori sia la loro selezione da un elenco, senza dover necessariamente conoscere la sintassi delle dichiarazioni corrispondenti. In questo modo è possibile prevenire molti piccoli errori come quelli di battitura durante la specifica di valori da una serie limitata. Tuttavia, per alcuni potrebbe risultare più rapido immettere direttamente le dichiarazioni. Per fare ciò, utilizzare la pagina Text.

Tutti gli elementi della libreria di tipi hanno una pagina di testo che visualizza la sintassi dell'elemento. Questa sintassi appare in un subset dell'IDL dell'Interface Definition Language di Microsoft. Qualsiasi modifica apportata in altre pagine dell'elemento viene riportata nella pagina di testo. Se si aggiunge del codice direttamente nella pagina Text, le modifiche vengono riportate nelle altre pagine del Type Library editor.

Il Type Library editor genera errori di sintassi se si aggiungono identificatori che non sono supportati attualmente dall'editor; l'editor supporta attualmente solamente quegli identificatori che fanno riferimento al supporto per le libreria di tipi (non supportano RPC né i costrutti utilizzati dal compilatore IDL di Microsoft per la generazione di codice sorgente C++ o per il supporto del marshaling).

Elementi del Type Library editor

A prima vista, l'interfaccia del Type Library editor può sembrare molto complicata. Ciò è dovuto al fatto che presenta informazioni su un elevato numero di elementi, ognuno dei quali ha caratteristiche proprie. Comunque, molte di queste caratteristiche sono comuni a tutti gli elementi. Ad esempio, ogni elemento (incluso la stessa libreria di tipi) ha le seguenti caratteristiche:

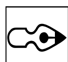
- AUn Name, utilizzato per descrivere l'elemento e per fare riferimento all'elemento nel codice.
- Un GUID (identificatore globalmente univoco), un valore globalmente univoco a 128-bit utilizzato da COM per identificare l'elemento. Questo dovrebbe essere sempre fornito per la stessa libreria di tipi, per le CoClass e per le interfacce. Altrimenti è facoltativo.

- Un Version number, per distingue tra più versioni dell'elemento. Questo è sempre facoltativo, ma dovrebbe essere fornito per CoClass e interfacce, poiché senza di esso alcuni strumenti non sono in grado di utilizzarli.
- Informazioni che collegano l'elemento a un argomento della Guida. Queste includono un valore Help String, Help Context o Help String Context. Help Context è utilizzato per un sistema di Guida tradizionale di Windows, in cui la libreria di tipi ha un file di Guida a sé stante. Help String Context si utilizza quando la Guida è fornita invece da una DLL separata. Help Context o Help String Context fa riferimento a un file di Guida o a una DLL specificate nella pagina Attributes della libreria di tipi. L'informazione è sempre è facoltativa.

Interfacce

Un' interfaccia descrive i metodi (e qualsiasi proprietà espressa come funzione 'get' e ' set') per un oggetto a cui si deve accedere tramite una tabella virtuale delle funzioni (Vtable). Se un'interfaccia è contrassegnata come duale, erediterà da *IDispatch*, e l'oggetto, tramite OLE automation, sarà in grado di eseguire il binding anticipato, di accedere alla vtable e di eseguire il binding in esecuzione. Per impostazione predefinita, la libreria di tipi contrassegna come duali tutte le interfacce aggiunte dall'utente.

Le interfacce possono essere membri assegnati: metodi e proprietà. Questi appaiono nel pannello Object list come figli del nodo dell'interfaccia. Le proprietà delle interfacce sono rappresentate dai metodi 'get' e 'set' utilizzati per leggere e scrivere i dati associati alla proprietà. Esse vengono rappresentate nella vista ad albero mediante delle icone particolari che ne indicano lo scopo.



Se una proprietà è specificata come Write By Reference, vuol dire che viene passata come puntatore piuttosto che per valore. Alcune applicazioni, come Visual Basic, usano Write By Reference, se presente, per ottimizzare le prestazioni. Per passare la proprietà solamente per riferimento invece che per valore, utilizzare il tipo di proprietà *By Reference Only*. Per passare la proprietà per riferimento e per valore, scegliere Read | Write | Write By Ref. Per richiamare questo menu, selezionare sulla barra strumenti la freccia accanto all'icona della proprietà.

Una volta aggiunte le proprietà o i metodi utilizzando il pulsante della barra strumenti o il menu contestuale del pannello Object list, se ne descrivono sintassi e attributi selezionando la proprietà o il metodo e utilizzando le pagine delle informazioni di tipo.

La pagina Attributes consente di assegnare alla proprietà o al metodo un nome e l'ID di smistamento (in modo da poterla chiamare utilizzando *IDispatch*). Per le proprietà, si deve assegnare anche un tipo. La firma della funzione viene creata utilizzando la pagina Parameters, in cui è possibile aggiungere, rimuovere e riordinare parametri, impostarne il tipo ed eventuali modificatori, e specificare i tipi restituiti della funzione.

Si noti che quando si assegnano proprietà e metodi a un'interfaccia, essi vengono assegnati implicitamente alla CoClass associata. Ciò è dovuto al fatto che il Type Library editor non consente di aggiungere direttamente proprietà e metodi a una CoClass.

Dispinterface

Le interfacce sono utilizzate più frequentemente delle dispinterface per descrivere proprietà e metodi di un oggetto. Le dispinterface sono accessibili solamente tramite binding dinamico, mentre le interfacce possono disporre del bind statico tramite una vtable.

È possibile aggiungere metodi e proprietà alle dispinterfaces secondo le stesse modalità usate per le interfacce. Comunque, quando si crea una proprietà per una dispinterface, non è possibile specificare il genere della funzione o i tipi dei parametri.

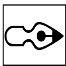
CoClass

Una CoClass descrive un oggetto COM univoco che implementa un o più interfacce. Quando si definisce una CoClass, è necessari specificare qual è l'interfaccia implementata predefinita dell'oggetto e, facoltativamente, quale dispinterface è la sorgente predefinita per gli eventi. Notare che nel Type Library editor non si aggiungono proprietà o metodi a una CoClass. Proprietà e metodi vengono esposti ai client dalle interfacce, che a loro volta vengono associate alle CoClass usando la pagina Implements.

Definizioni di tipo

Enumerativi, alias, record e union dichiarano tutti tipi che potrebbero essere quindi utilizzati altrove nella libreria di tipi.

Gli Enums sono costituiti da un elenco di costanti, ognuno delle quali deve essere numerica. L'input numerico normalmente è un intero in formato decimale o esadecimale. Per impostazione predefinita, il valore di base è zero. È possibile aggiungere costanti all'enumerativo selezionandolo nel pannello Object list e facendo clic sul pulsante Const sulla barra strumenti, oppure selezionando il comando New | Const nel menu contestuale del pannello Object list.



Si raccomanda vivamente di prevedere una serie di stringhe di guida per gli enumerativi, in modo da renderne chiaro il significato. Qui di seguito viene riportato un esempio di immissione di un tipo enumerativo per il pulsante di un mouse e include una stringa di guida per ogni elemento enumerativo.

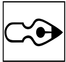
```
typedef enum TxMouseButton
{
   [helpstring("mbLeft”)]
   mbLeft = 0,
   [helpstring("mbRight)]
   mbRight = 1.
   [helpstring("mbMiddle)]
   mbMiddle = 2
} TxMouseButton;
```

Un alias crea un alias (typedef) per un tipo. È possibile utilizzare l'alias per definire tipi che si vogliono utilizzare in altre informazioni di tipo quali record o *union*. Associare l'alias con la relativa definizione di tipo impostando l'attributo Type nella pagina Attributes.

Un record è una struttura in stile C. Un record è costituito da una lista di membri della struttura o campi. Una *union* definisce una *union* in stile C. Analogamente a un record, una *union* è costituita da una lista di membri della struttura o campi. Comunque, diversamente dai membri dei record, ogni membro di una *union* occupa lo stesso indirizzo fisico, e pertanto è possibile memorizzare un solo valore logico.

Aggiungere i campi a un record o a una *union* selezionandolo nel pannello Object list e facendo clic sul pulsante del campo sulla barra strumenti, oppure facendo clic destro e scegliendo dal menu contestuale del pannello Object list l'opzione Field. Ogni campo ha un nome e un tipo che viene assegnato selezionando il campo e assegnandogli dei valori tramite la pagina Attributes. Record e *union* possono essere definite con un tag facoltativo, come con uno *struct* del C.

I membri possono essere di un qualsiasi tipo incorporato; è anche possibile specificarne uno usando un alias prima di definire il record.



C++Builder non supporta il marshaling delle parole chiave correlate a structs e *union*, come switch_type, first_is, last_is ecc.

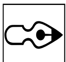
Moduli

Un modulo definisce un gruppo di funzioni, di solito una serie di punti d'ingresso nelle DLL. Un modulo può essere definito

- Specificando sulla pagina degli attributi la DLL che esso rappresenta.
- Aggiungendo metodi e costanti tramite la barra strumenti o il menu contestuale del pannello Object list. Per ogni metodo o costante, è necessario poi definirne gli attributi selezionandolo nel pannello Object list e impostandone i valori sulla pagina Attributes.

Per i metodi di modulo, si deve assegnare un nome e un punto di ingresso per la DLL utilizzando la pagina degli attributi. Dichiarare i parametri e il tipo restituito della funzione usando la pagina dei parametri.

Per costanti di modulo, utilizzare la pagina Attributes per specificare un nome, un tipo e un valore.



Il Type Library editor non genera nessuna dichiarazione o implementazione relativa a un modulo. La DLL specificata deve essere creata come progetto separato.

Uso del Type Library editor

Utilizzando l'editor delle librerie di tipi, è possibile creare nuove librerie di tipi o modificare quelle esistenti. Di solito, chi sviluppa applicazioni utilizza un wizard per creare gli oggetti esposti nella libreria di tipi, demandando a C++Builder la generazione automatica della libreria di tipi. Quindi, la libreria di tipi generata automaticamente viene aperta nel Type Library editor in modo da definire (o modificare) le interfacce, aggiungere definizioni di tipo, e così via.

Comunque, anche se non si utilizza un wizard per definire gli oggetti, è possibile utilizzare il Type Library editor per definire una nuova libreria di tipi. In questo caso, si devono creare tutte le classi di implementazione, poiché il Type Library editor non

genera codice per le CoClass che non sono state associate mediante un wizard a una libreria di tipi.

L'editor supporta un sottoinsieme di tipi validi in una libreria di tipi come descritto di seguito.

Gli argomenti finali in questa sezione descrivono come:

- Creare una nuova libreria di tipi
- Aprire una libreria di tipi esistente
- Aggiungere un'interfaccia alla libreria di tipi
- Modificare un'interfaccia
- Aggiungere proprietà e metodi alla libreria di tipi
- Aggiungere un CoClass alla libreria di tipi
- Aggiungere un'interfaccia a una CoClass
- Aggiungere un enumerativo alla libreria di tipi
- Aggiungere un alias alla libreria di tipi
- Aggiungere un record o una union alla libreria di tipi
- Aggiungere un modulo alla libreria di tipi
- Salvare e registrare le informazioni della libreria di tipi

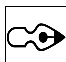
Tipi consentiti

Il Type Library editor supporta in una libreria di tipi i seguenti tipi IDL. La colonna Compatibile con Automation specifica se il tipo può essere utilizzato o meno da un'interfaccia i cui flag Automation o Dispinterface sono stati selezionati. Questi sono i tipi di cui COM è in grado di eseguire il marshaling automatico tramite la libreria di tipi.

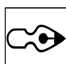
Tipo IDL	Tipo variant	Compatibile con Automation	Descrizione
short	VT_I2	Sì	Integer con segno a 2 byte
long	VT_I4	Sì	Integer con segno a 4 byte
single	VT_R4	Sì	Real a 4 byte
double	VT_R8	Sì	Real a 8 byte
CURRENCY	VT_CY	Sì	Valuta
DATE	VT_DATE	Sì	data
BSTR	VT_BSTR	Sì	Stringa binaria
IDispatch	VT_DISPATCH	Sì	Puntatore all'interfaccia IDispatch
SCODE	VT_ERROR	Sì	Codice di errore OLE
VARIANT_BOOL	VT_BOOL	Sì	True = –1, False = 0
VARIANT	VT_VARIANT	Sì	Puntatore a OLE Variant
IUnknown	VT_UNKNOWN	Sì	Puntatore all'interfaccia IUnknown
DECIMAL	VT_DECIMAL	Sì	virgola fissa a 16-byte
byte	VT_I1	No*	Integer con segno a 1 byte
unsigned char	VT_UI1	Sì	Integer senza segno a 1 byte
unsigned short	VT_UI2	No*	Integer senza segno a 2 byte

Tipo IDL	Tipo variant	Compatibile con Automation	Descrizione
unsigned long	VT_UI4	No*	Integer senza segno a 4 byte
__int64	VT_I8	No	Real con segno a 8 byte
uint64	VT_UI8	No	Real senza segno a 8 byte
int	VT_INT	No*	Integer dipendente dal sistema (Win32=Integer)
unsigned int	VT_UINT	No*	Integer senza segno dipendente dal sistema
void	VT_VOID	Sì	VOID in stile C
HRESULT	VT_HRESULT	No	Codice di errore a 32 bit
SAFEARRAY	VT_SAFEARRAY	Sì	OLE Safe Array
LPSTR	VT_LPSTR	No	Stringa terminata con null
LPWSTR	VT_LPWSTR	No	Wide string terminata con null

* Potrebbero essere compatibili con Automation con alcune applicazioni.



Il tipo unsigned char (VT_UI1) è compatibile con Automation, ma non ne è consentito l'uso in un Variant o OleVariant poiché molti server Automation non gestiscono correttamente questo valore.



Per i tipi consentiti nello sviluppo di applicazioni CORBA, consultare "Definizione delle interfacce dell'oggetto" a pagina 31-5.

A parte questi tipi IDL, in una definizione della libreria di tipi si può usare qualsiasi interfaccia e tipo definiti nella libreria o nelle librerie collegate.

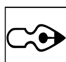
Il Type Library editor memorizza nel file generato di libreria di tipi (.TLB) le informazioni di tipo in formato binario.

SafeArray

COM richiede che gli array siano passati tramite un tipo di dati speciale noto come SafeArray. I SafeArray possono essere creati e distrutti chiamando speciali funzioni COM , e tutti gli elementi di un SafeArray devono essere tipi consentiti e compatibili con Automation.

Nel Type Library editor, un *SafeArray* deve specificare il tipo dei suoi elementi. Per esempio, la linea seguente prelevata dalla pagina Text dichiara un metodo con un parametro che è un *SafeArray*, il cui tipo dell'elemento è long:

```
HRESULT _stdcall  HighlightLines(SAFEARRAY(long) Lines);
```



Sebbene sia necessario specificare il tipo dell'elemento quando si dichiara un tipo *SafeArray* nel Type Library editor, la dichiarazione nella unit _TLB generata non indica il tipo dell'elemento.

Creazione di una nuova libreria di tipi

Potrebbe essere necessario creare una libreria di tipi indipendente di un particolare oggetto COM. Per esempio, si potrebbe voler definire una libreria di tipi che

contenga definizioni di tipo che si utilizzano in molte altre librerie di tipi. In questo casi si può creare una libreria di tipi di definizioni di base e aggiungerla alla pagina Uses di altre librerie di tipi.

È anche possibile creare anche una libreria di tipi per un oggetto non ancora implementato. Una volta che la libreria di tipi contiene la definizione dell'interfaccia, è possibile utilizzare il COM Object wizard per generare una CoClass e l'implementazione.

Per creare una nuova libreria di tipi:

1 Scegliere File|New|Other per aprire la finestra di dialogo New Items.
2 Scegliere la pagina ActiveX.
3 Selezionare l'icona Type Library.
4 Scegliere OK.

   Il Type Library editor appare con la richiesta di immettere un nome per la libreria di tipi.

5 Immettere il nome desiderato. Continuare ad aggiungere alla libreria di tipi gli elementi desiderati.

Apertura di una libreria di tipi esistente

Quando si usano i wizard per creare un controllo ActiveX, un oggetto Automation, ActiveForm, oggetti COM, oggetti transazionali, moduli dati remoti o moduli dati transazionali module, viene creata automaticamente una libreria di tipi con una unit di implementazione. Inoltre, si potrebbero avere librerie di tipi associate ad altri prodotti (server) disponibili nel sistema.

Per aprire una libreria di tipi che non fa parte del progetto corrente:

1 Scegliere il comando File|Open dal menu principale dell'IDE.
2 Nella finestra di dialogo Open, impostare la casella File Type a "type library.
3 Selezionare la libreria di tipi desiderata e scegliere Open.

Per aprire una libreria di tipi associata al progetto corrente:

1 Scegliere il comando View|Type Library.

A questo punto, è possibile aggiungere interfacce, CoClass e altri elementi della libreria di tipi, come enumerazioni, proprietà e metodi.



Quando si scrivono applicazioni client, non è necessario aprire la libreria di tipi. Occorre solamente la unit *Project*_TLB che il Type Library editor crea a partire da una libreria di tipi, e non la libreria di tipi. È possibile aggiungere direttamente questo file a un progetto client, oppure, se la libreria di tipi è registrata nel sistema, si può utilizzare la finestra di dialogo Import Type Library (comando *Project*|Import Type Library).

Aggiunta di un'interfaccia alla libreria di tipi

Per aggiungere un'interfaccia:

1 Nella barra strumenti fare clic sull'icona dell'interfaccia.

   Al pannello dell'elenco degli oggetti viene aggiunta un'interfaccia e compare la richiesta di inserire un nome.

2 Scrivere un nome per l'interfaccia.

La nuova interfaccia contiene gli attributi predefiniti che, se necessario, possono essere modificati.

È possibile aggiungere proprietà (rappresentate dalle funzioni getter/setter) e metodi adatti alla funzione dell'interfaccia.

Modifica di un'interfaccia con il Type Library editor

Esistono vari modi per modificare, una volta creata, un'interfaccia o una dispinterface.

- È possibile modificare gli attributi dell'interfaccia utilizzando la pagina Type information che contiene le informazioni che si desiderano modificare. Selezionare l'interfaccia nel pannello Object list ed utilizzate i controlli sulla pagina opportuna delle informazioni di tipo. Per esempio, si potrebbe voler modificare l'interfaccia del genitore utilizzando la pagina Attributes, oppure utilizzare la pagina Flags per indicare se l'interfaccia è duale o meno.
- È possibile modificare direttamente la dichiarazione dell'interfaccia selezionando l'interfaccia nel pannello Object list e modificando quindi le dichiarazioni sulla pagina Text.
- È possibile aggiungere proprietà e metodi all'interfaccia (vedere più avanti).
- È possibile modificare le proprietà e metodi già presenti nell'interfaccia modificandone le informazioni di tipo.
- È possibile associarla a una CoClass selezionando la CoClass nel pannello Object list, facendo clic destro sulla pagina Implements e scegliendo Insert Interface.

Se l'interfaccia è associata a una CoClass generata da un wizard, è possibile fare in modo che il Type Library editor applichi le modifiche al file di implementazione facendo clic sul pulsante Refresh sulla barra strumenti.

Aggiunta di proprietà e metodi ad una interface o dispinterface

Per aggiungere proprietà o metodi a una interface o dispinterface:

1 Selezionare l'interfaccia, e scegliere l'icona delle proprietà o dei metodi dalla barra strumenti. Se si sta aggiungendo una proprietà, si può fare clic direttamente sull'icona della proprietà per creare una proprietà di lettura/scrittura (con un getter e un setter), oppure fare clic sulla freccia verso il basso per visualizzare un menu di tipi della proprietà.

   Viene richiesta l'aggiunta di un nome; ciò fatto, i membri del metodo per accesso alla proprietà o il membro del metodo vengono aggiunti al pannello Object list.

2 Scrivere un nome per il membro.

Il nuovo membro contiene impostazioni predefinite, visibili sulle pagine Attributes, Parameters e Flags, che è possibile modificare per adattarle al membro. Per esempio, potrebbe essere necessario assegnare un tipo a una proprietà usando la pagina Attributes. Se si sta aggiungendo un metodo, potrebbe essere necessario specificarne i suoi parametri sulla pagina Parameters.

Come metodo alternativo, è possibile aggiungere proprietà e metodi scrivendo direttamente nella pagina Text e usando la sintassi IDL. Per esempio, nella pagina Text di un'interfaccia è possibile scrivere la seguente dichiarazione di proprietà:

```
[
  uuid(5FD36EEF-70E5-11D1-AA62-00C04FB16F42),
  version(1.0),
  dual,
  oleautomation
]
interface Interface1: IDispatch
{ // Add everything between the curly braces
[propget, id(0x00000002)]
  HRESULT _stdcall AutoSelect([out, retval] long Value );
  [propget, id(0x00000003)]
  HRESULT _stdcall AutoSize([out, retval] VARIANT_BOOL Value );
  [propput, id(0x00000003)]
  HRESULT _stdcall AutoSize([in] VARIANT_BOOL Value );
};
```

Dopo aver aggiunto i membri ad una pagina del membro interfaccia, i membri appaiono come elementi distinti nel pannello dell'elenco degli oggetti, ciascuno con le proprie pagine di attributi, flag e parametri. È possibile modificare ogni nuova proprietà o metodo selezionandolo nel pannello Object list e utilizzando queste pagine, oppure apportando le modifiche direttamente nella pagina Text.

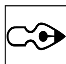
Se l'interfaccia è associata a una CoClass generata da un wizard, facendo clic sul pulsante Refresh sulla barra strumenti è possibile fare in modo che il Type Library editor applichi le modifiche al file di implementazione. Il Type Library editor aggiungerà nuovi metodi alla classe di implementazione per aggiornarla in base ai nuovi membri.In seguito, sarà possibile individuare i nuovi metodi nel codice sorgente della unit di implementazione e scriverne il corpo in modo da completare l'implementazione.

Aggiunta di CoClass alla libreria di tipi

Il modo più semplice per aggiungere una CoClass al progetto consiste nello scegliere dal menu principale nell'IDE il comando File|New|Other e utilizzare il wizard opportuno sulla pagina ActiveX o Multitier della finestra di dialogo New Items. Il vantaggio di questo metodo sta nel fatto che, oltre ad aggiungere la CoClass e la relativa interfaccia alla libreria di tipi, il wizard aggiunge una unit di implementazione, aggiorna il file di progetto includendo la nuova unit di implementazione, e aggiunge la nuova CoClass alla mappa degli oggetti nel file di progetto.

Se non si utilizza un wizard, comunque, è possibile creare una CoClass facendo clic sull'icona CoClass della barra strumenti e quindi specificandone gli attributi. Si vorrà assegnare un nome alla nuova CoClass (pagina Attributes), e utilizzare la pagina

Flags per inserire informazioni sulla CoClass, ad esempio, se è un oggetto applicazione, se rappresenta un controllo ActiveX, e così via.



Se si aggiunge una CoClass a una libreria di tipi utilizzando la barra strumenti invece di un wizard, si dovrà generare manualmente l'implementazione della CoClass e aggiornarla ogni volta che si modifica un elemento in una delle interfacce della CoClass. Quando si aggiunge manualmente l'implementazione di una CoClass, accertarsi di aggiungere la nuova CoClass alla mappa degli oggetti nel file di progetto.

Non è possibile aggiungere membri a un CoClass in modo diretto. I membri vengono aggiunti in modo implicito quando si aggiunge un'interfaccia alla CoClass.

Aggiunta di un'interfaccia a una CoClass

Le CoClass sono definite dalle interfacce che esse rendono disponibili ai client. Benché sia possibile aggiungere alla classe di implementazione di una CoClass tutte le proprietà e metodi desiderati, i client possono vedere solo quelle proprietà e quei metodi che vengono esposti dalle interfacce associate con la CoClass.

Per associare un'interfaccia con una CoClass, fare clic destro nella pagina Implements della classe e scegliere Insert Interface in modo da visualizzare un elenco di interfacce tra cui effettuare la scelta. L'elenco include le interfacce definite nella libreria di tipi corrente e quelle definite in una qualsiasi libreria di tipi referenziata nella libreria di tipi corrente. Scegliere l'interfaccia che dovrà essere implementata dalla classe. L'interfaccia viene aggiunta alla pagina con il relativo GUID e gli altri attributi.

Se la CoClass è stata generato usando un wizard, il Type Library editor aggiorna automaticamente la classe di implementazione includendo lo scheletro di base dei metodi per quei metodi (inclusi i metodi per l'accesso alle proprietà) di tutte le interfacce aggiunte in questo modo.

Aggiunta di una Enumeration alla libreria di tipi

Per aggiungere enumerativi ad una libreria di tipi:

1 Sulla barra strumenti fare clic sull'icona enum.

   Un tipo enum viene aggiunto al pannello Object list e compare la richiesta di inserire un nome.

2 Scrivere un nome per l'enumerativo.

   Il nuovo enum è vuoto e contiene gli attributi predefiniti nella pagina Attributes dove è possibile modificarli.

Aggiungere i valori alla enumerazione facendo clic sul pulsante New Const. Quindi selezionare ciascun valore enumerato e assegnargli il nome (e possibilmente un valore) usando la pagina degli attributi.

Una volta aggiunto un enumerativo, il nuovo tipo è disponibile per l'utilizzo da parte della libreria di tipi o di qualsiasi altra libreria di tipi che contenga un riferimento ad esso nella pagina Uses. Ad esempio, è possibile utilizzare l'enumerativo come tipo di una proprietà o di un parametro.

Aggiunta di un alias alla libreria di tipi

Per aggiungere un alias a una libreria di tipi:

1 Sulla barra strumenti, fare clic su sull'icona dell'alias.

   Un tipo alias verrà aggiunto al pannello Object list e apparirà la richiesta di aggiungere un nome.

2 Scrivere un nome per l'alias.

   Per impostazione predefinita, il nuovo alias è di tipo long.Utilizzare la pagina Attributes per modificarlo nel tipo che si vuole sia rappresentato dall'alias.

Una volta aggiunto un alias, il nuovo tipo è disponibile per l'utilizzo da parte della libreria di tipi o di qualsiasi altra libreria di tipi che contenga un riferimento ad esso nella pagina Uses. Ad esempio, è possibile utilizzare l'alias come tipo di una proprietà o di un parametro.

Aggiunta di un record o di una union alla libreria di tipi

Per aggiungere un record o una union a una libreria di tipi:

1 Sulla barra strumenti, fare clic sull'icona del record o su quella della union.

   L'elemento di tipo selezionato verrà aggiunto al pannello Object list e apparirà la richiesta di aggiungere un nome.

2 Scrivere un nome per il record o per la unione.

   A questo punto, il nuovo record o la union non contengono campi.

3 Dopo aver selezionato il record o la union nel pannello Object list, fare clic sull'icona Field nella barra strumenti. Specificare nome e tipo di campo utilizzando la pagina Attributes.

4 Ripetere il passo 3 per tutti i campi necessari.

Una volta definito un record o una union, il nuovo tipo è disponibile per l'utilizzo da parte della libreria di tipi o di qualsiasi altra libreria di tipi che contenga un riferimento ad esso nella pagina Uses. Ad esempio, è possibile utilizzare il record o la union come tipo di una proprietà o di un parametro.

Aggiunta di un modulo alla libreria di tipi

Per aggiungere un modulo a una libreria di tipi:

1 Sulla barra strumenti, fare clic su sull'icona del modulo.

   Il modulo scelto verrà aggiunto al pannello di Object list e e apparirà la richiesta di aggiungere un nome.

2 Scrivere un nome per il modulo.

3 Sulla pagina Attributes, specificare il nome della DLL di cui il modulo ne rappresenta il punto d'ingresso.

4 Aggiungere tutti i metodo dalla DLL specificata al punto 3, facendo clic sull'icona Method nella barra strumenti, e utilizzando successivamente le pagine degli attributi per descrivere il metodo.

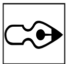
5 Aggiungere tutte le costanti che si vuole vengano definite dal modulo facendo clic su sull'icona Const sulla barra strumenti. Per ogni costante, specificare nome, tipo e valore.

Salvataggio e registrazione delle informazioni della libreria di tipi

Dopo aver modificato la libreria di tipi, sarà necessario salvare e registrare le informazioni relative.

Il salvataggio di una libreria di tipi aggiorna automaticamente:

- Il file binario della libreria di tipi (estensione .tlb).
- La unit *Project*_TLB che ne rappresenta i contenuti
- Il codice dell'implementazione di tutte le CoClass generate da un wizard.



La libreria di tipi viene registrata come file binario separato (.TLB), ma viene anche eseguito il link nel server (.EXE, DLL o .OCX).

Il Type Library editor fornisce le opzioni per memorizzare le informazioni della libreria di tipi. Il modo prescelto per questa operazione dipende dallo stadio a cui si è giunti nell'implementazione della libreria di tipi:

- Save, per salvare entrambe le unit .TLB e la unit *Project*_TLB sul disco.
- Refresh, per aggiornare le unit della libreria di tipi solo in memoria.
- Register, per aggiungere un elemento della libreria di tipi nel registro di sistema di Windows. L'operazione viene eseguita automaticamente se il server con cui è associato il file .TLB è esso stesso registrato.
- Export, per salvare un file .IDL contenente le definizione dei tipi e delle interfacce secondo la sintassi IDL.

Tutti i metodi precedenti eseguono un controllo della sintassi. Quando si aggiorna, registra o salva la libreria di tipi, C++Builder aggiorna automaticamente la unit di implementazione di ogni CoClasses che è stata creata usando un wizard.

Salvataggio di una libreria di tipi

Il salvataggio di una libreria di tipi

- Esegue un controllo sintattico e di validità.
- Salva le informazioni in un file .TLB.
- Salva le informazioni nella unit *Project*_TLB.
- Notifica al gestore del modulo IDE di aggiornare l'implementazione nel caso la libreria di tipi sia associata a una CoClass che è stata generata da un wizard.

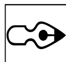
Per salvare la libreria di tipi, scegliere File|Save dal menu principale di C++Builder.

Aggiornamento della libreria di tipi

L'aggiornamento della libreria di tipi

- Esegue il controllo della sintassi.
- Rigenera le unit della libreria di tipi di C++Builder solo in memoria. Non salva alcun file su disco.
- Notifica al gestore del modulo IDE di aggiornare l'implementazione nel caso la libreria di tipi sia associata a una CoClass che è stata generata da un wizard.

Per aggiornare la libreria di tipi scegliere l'icona Refresh nella barra strumenti del Type Library editor.



Se sono stati ridenominati degli elementi nella libreria di tipi, l'aggiornamento dell'implementazione può creare elementi duplicati. In questo caso, occorre spostare il codice nel punto corretto e cancellare i doppioni. Analogamente, se si cancellano elementi nella libreria di tipi, l'aggiornamento dell'implementazione non li rimuove dalle CoClasses (partendo dal presupposto che si stia rimuovendo solo la loro visibilità ai client). Se non sono più necessari, si dovrà cancellare manualmente questi elementi nella unit di implementazione.

Registrazione della libreria di tipi

Di solito, non è necessario registrare esplicitamente una libreria di tipi in quanto essa si registra automaticamente al momento della registrazione dell'applicazione COM server (consultare "Registrazione di un oggetto COM" a pagina 41-17). Comunque, quando si crea una libreria di tipi utilizzando il Type Library wizard, essa non è associata con un oggetto server. In questo caso, è possibile registrare la libreria di tipi utilizzando direttamente la barra strumenti.

La registrazione della libreria di tipi:

- Esegue un controllo della sintassi
- Aggiunge un elemento per la libreria di tipi nel registro di Windows

Per registrare la libreria di tipi, scegliere l'icona Register nella barra strumenti del Type Library editor.

Esportazione di un file IDL

L'esportazione di una libreria di tipi:

- Esegue il controllo della sintassi.
- Crea un file IDL contenente le dichiarazioni delle informazioni di tipo. Questo file descrive le informazioni di tipo secondo la sintassi IDL di Microsoft.

Per esportare la libreria di tipi, scegliere l'icona Export nella barra strumenti del Type Library editor.

Distribuzione delle librerie di tipi

Per impostazione predefinita, se una libreria di tipi è stata creata come parte di un progetto di un server ActiveX o Automation, essa viene collegata automaticamente al file .DLL, .OCX o EXE sotto forma di risorsa.

È possibile, tuttavia, distribuire l'applicazione con la libreria di tipi sotto forma di file .TLB distinto, in quanto C++Builder, se lo si preferisce, è in grado di eseguire la manutenzione della libreria di tipi.

Storicamente, le librerie di tipi per le applicazioni Automation erano memorizzate come fili distinti con estensione .TLB. Ora, la tipiche applicazioni di Automation compilano le librerie di tipi direttamente nei file .OCX o .EXE. Il sistema operativo presume che la libreria di tipi sia la prima risorsa nel file eseguibile (.OCX o .EXE).

Quando si rendono disponibili agli sviluppatori di applicazione altre librerie di tipi che non siano quella del progetto principale, esse possono essere in una qualsiasi delle seguenti forme:

- Una Risorsa. Questa risorsa avrà il tipo TYPELIB e un intero ID. Se si sceglie di costruire librerie di tipi con un compilatore di risorse, occorre dichiararlo nel file risorsa (.RC) nel modo seguente:

  ```
  1 typelib mylib1.tlb
  2 typelib mylib2.tlb
  ```

  In una libreria ActiveX ci possono essere più risorse di libreria di tipi. Gli sviluppatori di applicazioni usano il compilatore di risorse per aggiungere il file .TLB alla propria libreria ActiveX.

- File binari indipendenti. Il file .TLB generato dal Type Library editor è un file binario.

Capitolo

40

Creazione di client COM

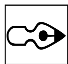
I client COM sono applicazioni che fanno uso di un oggetto COM implementato da un'altra applicazione o da una libreria. I tipi più comuni sono applicazioni che controllano un Automation server (Automation controller) e applicazioni che ospitano un controllo ActiveX (ActiveX container).

A prima vista questi due tipi di client COM sono molto diversi: L'Automation controller tipico lancia un server EXE esterno e impartisce dei comandi per far sì che quel server esegua dei compiti per suo conto. L'Automation server di solito è non visuale e out-of-process. Il tipico client ActiveX, d'altro canto, ospita un controllo visuale, utilizzandolo in modo molto simile a come si utilizza un qualsiasi controllo sulla Component palette. I server ActiveX sono sempre server in-process.

Comunque, le operazioni per la scrittura questi due tipi di client COM sono notevolmente simili: L'applicazione client ottiene un'interfaccia per l'oggetto server e ne usa proprietà e metodi. C++Builder semplifica tutto ciò consentendo di avvolgere la CoClass del server in un componente sul client, che sarà possibile installare anche sulla Component palette. Esempi di tali wrapper di componenti appaiono su due pagine della Component palette: esempi di wrapper ActiveX appaiono sulla pagina ActiveX e esempi di oggetti Automation appaiono sulla pagina Servers.

Quando si scrive un client COM, è necessario comprendere l' interfaccia che il server espone ai client, esattamente come si devono comprendere proprietà e metodi di un componente della Component palette per poterlo utilizzarlo nelle applicazioni. Questa interfaccia (o insieme di interfacce) è determinata dall'applicazione server, e di solito resa pubblica in una libreria di tipi. Per informazioni specifiche sulle interfacce rese pubbliche di una particolare applicazione server, si dovrebbe consultare la documentazione di quella applicazione.

Anche se si sceglie di non avvolgere un oggetto server in un wrapper di componenti e di non installarlo sulla Component palette, occorre rendere disponibile all'applicazione la sua definizione di interfaccia. Per fare ciò è possibile importare le informazioni di tipo del server.



È anche possibile richiedere direttamente le informazioni di tipo utilizzano le API di COM, ma C++Builder non fornisce un particolare supporto adatto allo scopo.

Alcune vecchie tecnologie COM, come OLE (Object Linking and Embedding) non forniscono informazioni di tipo in una libreria di tipi. Si appoggiano, invece, ad un set standard di interfacce predefinite. L'argomento è trattato nella sezione "Creazione di client per server che non hanno una libreria di tipi" a pagina 40-17.

Importazione di informazioni della libreria di tipi

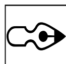
Per rendere disponibili all'applicazione client le informazioni sul server COM occorre importare le informazioni sul server memorizzate nella libreria di tipi del server. L'applicazione potrà utilizzare quindi le classi generate risultanti per controllare l'oggetto server.

Esistono due modi per importare le informazioni della libreria di tipi:

- È possibile utilizzare la finestra di dialogo Import Type Library per importare tutte le informazioni disponibili sui tipi, sugli oggetti e sulle interfacce del server. Questo è il metodo più generale, in quanto consente di importare informazioni da qualsiasi libreria di tipi e può generare, all'occorrenza, wrapper di componenti per tutte le CoClasses creabili, incluse nella libreria di tipi e non contrassegnate come Hidden, Restricted o PreDeclID.
- Se si sta eseguendo l'importazione dalla libreria di tipi di un controllo ActiveX, è possibile utilizzare la finestra di dialogo Import ActiveX. L'operazione importa le stesse informazioni di tipo, ma crea wrapper di componenti solamente per le CoClass che rappresentano controlli ActiveX.
- È possibile utilizzare il programma di utilità della riga comandi tlibimp.exe che dispone di ulteriori opzioni di configurazione non disponibili dall'interno dell'IDE.
- Una libreria di tipi generata utilizzando un wizard viene importata automaticamente utilizzando lo stesso meccanismo utilizzato dalla voce di menu Import type library.

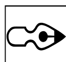
Indipendentemente dal metodo scelto per importare informazioni della libreria di tipi, la finestra di dialogo risultante crea una unit di nome *TypeLibName*_TLB, dove *TypeLibName* è il nome della libreria di tipi. Questo file contiene le dichiarazioni per le classi, i tipi e le interfacce definite nella libreria di tipi. Includendolo nel progetto, quelle definizioni saranno disponibili all'applicazione, rendendo così possibile creare oggetti e chiamarne le interfacce. Questo file può essere ricreato dall'IDE di tanto in tanto; pertanto, non è consigliabile apportare manualmente modifiche al file.

Oltre ad aggiungere le definizioni di tipo alla unit *TypeLibName*_TLB, la finestra di dialogo può creare anche wrapper della classe VCL per tutte le CoClass definite nella libreria di tipi, che metterà in una unit separata di nome *TypeLibName*_OCX. Quando si utilizza la finestra di dialogo Import Type Library, la creazione di questi wrapper è facoltativa. Quando si utilizza la finestra di dialogo Import ActiveX, questi wrapper vengono sempre generati per tutte le CoClass che rappresentano controlli.



Se si stanno generando wrapper di componenti, la finestra di dialogo di importazione genera la unit *TypeLibName*_OCX, ma non la aggiunge al progetto. (Aggiunge solamente la unit *TypeLibName*_TLB). È possibile aggiungere esplicitamente al progetto la unit *TypeLibName*_OCX scegliendo l'opzione Project | Add to Project.

I wrapper di classe generati rappresentano le CoClass all'applicazione, ed espongono le proprietà e i metodi delle sue interfacce. Se una CoClass supporta le interfacce per la generazione di eventi (*IConnectionPointContainer* e *IConnectionPoint*), il wrapper della classe VCL crea un convogliatore di eventi in modo che l'assegnazione dei gestori di evento risulti semplice come per qualsiasi altro componente. Se nella finestra di dialogo si sceglie di installare le classi VCL generate nella Component palette, è possibile utilizzare l'Object Inspector per assegnare valori di proprietà e gestori di evento.



La finestra di dialogo Import Type Library non crea wrapper di classe per oggetti evento di COM+. Per scrivere un client che risponda agli eventi generati da un oggetto evento di COM+, si deve creare da programma il convogliatore dell'evento. Questo procedimento è descritto in "Gestione di eventi COM+" a pagina 40-16.

Per maggiori informazioni sul codice generato durante l'importazione di una libreria di tipi, consultare "Codice generato durante l'importazione delle informazioni di una libreria di tipi" a pagina 40-5.

Uso della finestra di dialogo Import Type Library

Per importare una libreria di tip:

1 Scegliere il comando Project | Import Type Library.

2 Selezionare dall'elenco la libreria di tipi.

   La finestra di dialogo elenca tutte le librerie di tipi registrate nel sistema. Se la libreria non si trova nell'elenco, scegliere il pulsante Add, cercare e selezionare il file di libreria di tipi, scegliere OK. In questo modo si registra la libreria di tipi, rendendola pertanto disponibile. Quindi ripetere il punto 2. Si noti che la libreria di tipi potrebbe essere un file di libreria indipendente (.tlb, .olb), oppure un server che mette a disposizione una libreria di tipi (.dll, .ocx, .exe).

3 Se si vuole generare un componente VCL che avvolga una CoClass nella libreria di tipi, selezionare l'opzione Generate Component Wrapper. Se non si vuole generare il componente, si può sempre utilizzare la CoClass utilizzando le definizioni nella unit *TypeLibName*_TLB. Tuttavia, sarà necessario scrivere personalmente le chiamate per creare l'oggetto server e, se occorre, impostare un convogliatore di eventi.

   La finestra di dialogo Import Type importa solo le CoClass il cui flag CanCreate è impostato e i cui flag Hidden, Restricted o PreDeclID non sono impostati. Questi flag possono essere ridefiniti utilizzando il programma di utilità della riga comandi tlibimp.exe.

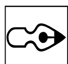
4 Se non si vuole installare nella Component palette il wrapper di componenti generato, scegliere *Create* Unit. L'operazione genera la unit *TypeLibName*_TLB e, se

al punto 3 è stata selezionata l'opzione Generate Component Wrapper, la unit *TypeLibName*_OCX. Questo conclude anche le operazioni nella finestra di dialogo Import Type Library.

5 Se si vuole installare nella Component palette il wrapper di componenti generato, selezionare la pagina della Component palette su cui dovrà risiedere questo componente e scegliere l'opzione Install. L'operazione genera le unit *TypeLibName*_TLB e *TypeLibName*_OCX, analogamente al pulsante *Create* Unit, quindi visualizza la finestra di dialogo Install component che consente di specificare il package in cui collocare i componenti (sia esso un package esistente o uno nuovo). Il pulsante risulta disattivato (grigio) nel caso non sia possibile creare un componente per la libreria di tipi.

Appena si esce dalla finestra di dialogo Import Type Library, le nuove unit *TypeLibName*_TLB e *TypeLibName*_OCX appariranno nella directory specificata dal controllo Unit dir name. La unit *TypeLibName*_TLB contiene le dichiarazioni per gli elementi definiti nella liberia di tipi. La unit *TypeLibName*_OCX contiene il wrapper di componenti generato, nel caso sia stata selezionata l'opzione Generate Component Wrapper.

Inoltre, se è stato installato il wrapper di componenti generato, a questo punto nella Component Palette risiede un oggetto server descritto dalla libreria di tipi. È possibile utilizzare l'Object Inspector per impostare le proprietà o per scrivere un gestore di evento per il server. Se si aggiunge il componente a una scheda o ad un modulo dati, in fase di progettazione è possibile farvi clic destro per visualizzarne la pagina di proprietà (nel caso la supporti).



La pagina Servers della Component palette contiene diversi esempi di Automation server importati nel modo appena descritto.

Uso della finestra di dialogo Import ActiveX

Per importare un controllo ActiveX:

1 Scegliere l'opzione Component|Import ActiveX Control.

2 Selezionare dall'elenco la libreria di tipi.

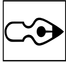
   La finestra di dialogo elenca tutte le librerie di tipi registrate nel sistema che definiscono controlli ActiveX. (Questo è un sottoinsieme delle librerie elencate nella finestra di dialogo Import Type Library). Se la libreria non si trova nell'elenco, scegliere il pulsante Add, cercare e selezionare il file di libreria di tipi, scegliere OK. In questo modo si registra la libreria di tipi, rendendola pertanto disponibile. Quindi ripetere il punto 2. Si noti che la libreria di tipi potrebbe essere un file di libreria indipendente (.tlb, .olb), oppure un server ActiveX (.dll, .ocx).

3 Se non si vuole installare nella Component palette il controllo ActiveX, scegliere *Create* Unit. L'operazione genera la unit *TypeLibName*_TLB e la unit *TypeLibName*_OCX. Questo conclude anche le operazioni nella finestra di dialogo Import ActiveX.

4 Se si vuole installare nella Component palette il controllo ActiveX, selezionare la pagina della Component palette su cui dovrà risiedere questo componente e

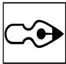
scegliere l'opzione Install. L'operazione genera le unit *TypeLibName*_TLB e *TypeLibName*_OCX, analogamente al pulsante *Create* Unit, quindi visualizza la finestra di dialogo Install component che consente di specificare il package in cui collocare i componenti (sia esso un package esistente o uno nuovo).

Appena si esce dalla finestra di dialogo Import ActiveX, le nuove unit *TypeLibName*_TLB e *TypeLibName*_OCX appariranno nella directory specificata dal controllo Unit dir name. La unit *TypeLibName*_TLB contiene le dichiarazioni per gli elementi definiti nella liberia di tipi. La unit *TypeLibName*_OCX contiene il wrapper di componenti generato per il controllo ActiveX.



A differenza della finestra di dialogo Import Type Library in cui l'opzione è facoltativa, la finestra di dialogo Import ActiveX genera sempre un wrapper di componenti. Il motivo dipende dal fatto che, essendo un controllo visuale, un controllo ActiveX necessita del supporto aggiuntivo del wrapper di componenti per poter essere collocato sulle schede VCL.

Inoltre, se il componente wrapper generato è stato installato, a questo punto nella Component Palette risiede un controllo ActiveX. È possibile utilizzare l'Object Inspector per impostare le proprietà o per scrivere i gestori di evento del controllo. Se si aggiunge il controllo a una scheda o a un modulo dati, in fase di progettazione è possibile farvi clic destro per visualizzare la pagina di proprietà (nel caso la supporti).



La pagina ActiveX della Component palette contiene diversi esempi di controlli ActiveX importati nel modo appena descritto.

Codice generato durante l'importazione delle informazioni di una libreria di tipi

Dopo aver importato una libreria di tipi, è possibile visualizzare la unit *TypeLibName*_TLB generata. Il file sorgente di questa unit definisce le costanti che assegnano nomi simbolici ai GUID della libreria di tipi e alle sue interfacce e CoClass. I nomi delle costanti sono generati come segue:

- Il GUID per la libreria di tipi assume il formato LIBID_*TypeLibName*, in cui *TypeLibName* è il nome della libreria di tipi.
- Il GUID per un'interfaccia assume il formato IID_*InterfaceName*, in cui *InterfaceName* è il nome dell'interfaccia.
- Il GUID per una dispinterface assume il formato DIID_*InterfaceName*, in cui *InterfaceName* è il nome della dispinterface.
- Il GUID per una CoClass assume il formato CLSID_*ClassName*, in cui *ClassName* è il nome della CoClass.

Facendo clic destro nel file sorgente e scegliendo l'opzione Open Source/Header file, si vedranno le definizioni seguenti:

- Dichiarazioni per le CoClasses nella libreria di tipi. Queste mappano ogni CoClass alla rispettiva interfaccia predefinita. Inoltre, viene dichiarato un wrapper per

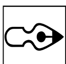
ciascuna CoClass, utilizzando il modello *TComInterface*. Il nome di questo wrapper assume il formato CoClassNamePtr.

- Dichiarazioni per le interfacce e le dispinterface nella libreria di tipi.
- Dichiarazioni di wrapper di classe per le interfacce e dispinterface. Per le interfacce, il wrapper della classe utilizza il modello *TComInterface*, e il nome assume il formato TComInterfaceName. Per le dispinterface, il wrapper della classe utilizza il modello *TAutoDriver*, e il cui nome assume il formato InterfaceNameDisp.
- Dichiarazioni per una classe creatore per ciascuna CoClass la cui interfaccia predefinita supporti il binding mediante VTable. La classe creatore ha due metodi statici, *Create* e *CreateRemote* che possono essere utilizzati per istanziare la CoClass localmente (*Create*) oppure in remoto (*CreateRemote*). Questi metodi restituiscono il wrapper della classe per l'interfaccia predefinita della CoClass (definita in *TypeLibName*_TLB.h utilizza il modello *TComInterface*.
- Un wrapper della classe proxy per ogni interfaccia evento.Questo wrapper di classe ha un nome che assume il formato TEvents_*CoClassName*, in cui *CoClassName* è il nome della CoClass che genera gli eventi. La classe proxy contiene un elenco di tutti i convogliatori degli eventi dei client e richiama il metodo opportuno su tutti i convogliatori di evento appropriati non appena l'oggetto server scatena tali eventi.

Queste dichiarazioni mettono a disposizione tutto ciò che occorre per creare istanze della CoClass e accedere alla sua interfaccia. Tutto ciò che occorre fare è includere il file *TypeLibName*_TLB.h generato nella unit in cui si vuole eseguire il bind a una CoClass e chiamarne le interfacce.



I wrapper delle classi generati utilizzano un oggetto statico (*TInitOleT*) per gestire l'inizializzazione COM. Questo può essere un problema se si stanno creando oggetti da più thread, in quanto COM viene inizializzato solamente sul primo thread che si connette a un server. È necessario creare esplicitamente un'istanza separata di *TInitOleT* o di *TInitOle* su tutti gli altri thread che si connettono a un server COM.

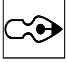


Quando si utilizza il Type Library editor o il programma di utilità TLIBIMP viene generata anche la unit *TypeLibName*_TLB.

Se si vuole utilizzare un controllo ActiveX, oltre alle dichiarazioni descritte in precedenza, è necessario avere anche il wrapper della VCL generato. Il wrapper della VCL si occupa delle problematiche di gestione della finestra del controllo. Nella finestra di dialogo Import Type Library dovrebbe essere stato generato anche un wrapper VCL per le altre CoClass. Questi wrapper VCL semplificano il compito di creare oggetti server e di chiamare i rispettivi metodi. Se ne raccomanda l'uso specialmente se si desidera che l'applicazione client risponda agli eventi.

Le dichiarazioni per i wrapper VCL generati appaiono nella unit *TypeLibName*_OCX. I wrapper di componenti per i controlli ActiveX derivano da *TOleControl*. I wrapper di componenti per gli oggetti Automation derivano da *TOleServer*. Il wrapper di componente generato aggiunge le proprietà, gli eventi e i metodi esposti dall'interfaccia della CoClass. È possibile utilizzare questo componente come qualsiasi altro componente VCL.


Non si dovrebbero modificare le unit *TypeLibName*_TLB o *TypeLibName*_OCX generate. Esse vengono rigenerate ogni volta che si aggiorna la libreria di tipi e pertanto tutte le modifiche vengono sovrascritte.


Per informazioni più aggiornate sul codice generato, fare riferimento ai commenti nella unit *TypeLibName*_TLB e nel file utilcls.h generati automaticamente.

Controllo di un oggetto importato

Dopo aver importato le informazioni della libreria di tipi, si può iniziare a programmare con gli oggetti importati. Il come procedere dipende in parte dagli oggetti e in parte dal fatto che si sia scelto o meno di creare wrapper di componenti.

Uso di wrapper di componenti

Se è stato generato un wrapper di componente per l'oggetto server, la scrittura dell'applicazione client COM non è molto diversa dalla scrittura di una qualsiasi altra applicazione contenente componenti VCL. Le proprietà, i metodi e gli eventi dell'oggetto server sono già incapsulati nel componente VCL. Sarà sufficiente assegnare solamente i gestori di evento, impostare i valori delle proprietà e i metodi di chiamata.

Per utilizzare le proprietà, i metodi e gli eventi dell'oggetto server, consultare la documentazione del server. Se possibile, il wrapper del componente fornisce automaticamente un'interfaccia duale. C++Builder determina la struttura della VTable dalle informazioni nella libreria di tipi.

Inoltre, il nuovo componente eredita alcune importanti proprietà e metodi dalla sua classe base.

Wrapper ActiveX

Quando si ospitano controlli ActiveX, si dovrebbe utilizzare sempre un wrapper di componente in quanto esso integra la finestra del controllo nel framework della VCL.

Le proprietà e i metodi che un controllo ActiveX eredita da *TOleControl* consentono di accedere all'interfaccia associata o di ottenere informazioni sul controllo. La maggior parte delle applicazioni, comunque, non ha bisogno di utilizzarli. Per contro, si utilizzerà il controllo importato secondo le stesse modalità usate per un qualsiasi altro controllo VCL.

Di solito, i controlli ActiveX dispongono di una pagina di proprietà che consente di impostarne le proprietà. Le pagine di proprietà sono simili agli editor di componenti che alcuni componenti visualizzano quando si fa doppio clic su di essi nel Form Designer. Per visualizzare la pagina di proprietà di un controllo ActiveX, fare clic destro e scegliere l'opzione Properties.

Il modo in cui si utilizza la maggior parte dei controlli ActiveX importati è determinato dall'applicazione server. Comunque, i controlli ActiveX utilizzano una serie standard di notifiche quando rappresentano i dati di un campo di database. Per

informazioni su come utilizzare questi controlli ActiveX, consultare "Uso di controlli ActiveX associati ai dati" a pagina 40-9 Il modo in cui si utilizza la maggior parte dei controlli ActiveX importati è determinato dall'applicazione server. Comunque, i controlli ActiveX utilizzano una serie standard di notifiche quando rappresentano i dati di un campo di database. Per informazioni su come utilizzare questi controlli ActiveX, consultare.

Wrapper di oggetti Automation

I wrapper degli oggetti Automation consentono di controllare il modo in cui si desidera stabilire la connessione con l'oggetto server:

- La proprietà *ConnectKind* indica se il server è locale o remoto e se si desidera effettuare la connessione a un server che è già in esecuzione o ad una nuova istanza che dovrebbe essere lanciata. Quando si effettua la connessione a un server remoto, si deve specificare il nome della macchina utilizzando la proprietà *RemoteMachineName*.
- Una volta specificato il tipo di connessione con *ConnectKind*, esistono tre modi per connettere il componente al server:
  - Si può eseguire la connessione al server in modo esplicito chiamando il metodo *Connect* del componente.
  - Si può dire al componente di connettersi automaticamente all'avvio dell'applicazione impostando la proprietà *AutoConnect* a **true**.
  - Non è necessario eseguire esplicitamente la connessione al server. Il componente stabilisce automaticamente una connessione non appena si utilizza una proprietà o un metodo del server utilizzando il componente.

Le modalità di chiamata dei metodi o di accesso alle proprietà sono analoghe a quelle utilizzate per qualsiasi altro componente:

```
TServerComponent1->DoSomething();
```

La gestione degli eventi è semplice, poiché è possibile utilizzare l'Object Inspector per scrivere i gestori di evento. Si noti, tuttavia, che il gestore di evento del componente potrebbe avere parametri leggermente diversi da quelli definiti per l'evento nella libreria di tipi. Di solito, quei parametri che risultano essere puntatori alle interfacce vengono avviluppati utilizzando il modello *TComInterface* e non appaiono come tipici puntatori alle interfacce. Il nome del wrapper di interfaccia risultante assume il formato *InterfaceNamePtr*.

Per esempio, il codice seguente mostra un gestore di evento per l'evento OnNewWorkBook di ExcelApplication. Il gestore di evento ha un parametro che fornisce l'interfaccia di un'altra CoClass (ExcelWorkbook). Comunque, l'interfaccia non viene passata come un puntatore all'interfaccia di ExcelWorkbook, ma piuttosto come un oggetto *ExcelWorkbookPtr*.

```
void _fastcall TForm1::XLappNewWorkbook(TObject *Sender, ExcelWorkbookPtr Wb)
{
   ExcelWorkbook1->ConnectTo(Wb);
}
```

In questo esempio, il gestore di evento assegna il workbook a un componente ExcelWorkbook, (ExcelWorkbook1). L'esempio mostra come connettere un wrapper di componente a un'interfaccia esistente utilizzando il metodo *ConnectTo*. Il metodo *ConnectTo* viene aggiunto al codice generato per il wrapper del componente.

I server che hanno un oggetto applicazione espongono su quell'oggetto un metodo Quit per consentire ai client di terminare la connessione. Quit espone di solito funzionalità che sono equivalenti all'utilizzo del menu File per terminare l'applicazione. Il codice per chiamare il metodo Quit viene generato nel metodo *Disconnect* del componente. Se è possibile chiamare il metodo Quit senza parametri, il wrapper del componente ha anche una proprietà *AutoQuit*. *AutoQuit* fa in modo che il controller chiami il metodo Quit nel momento in cui il componente viene liberato. Se si vuole effettuare la disconnessione in un altro momento, oppure se il metodo Quit richiede dei parametri, è necessario chiamarlo esplicitamente. Quit appare nel componente generato come un metodo pubblico.

Uso di controlli ActiveX associati ai dati

Quando si utilizza un controllo ActiveX associato ai dati in un'applicazione C++Builder, è necessario associarlo al database di cui ne rappresenta i dati. Per fare ciò, è necessario un componente data source, esattamente come è necessario un data source quando si utilizza un qualsiasi controllo VCL associato ai dati.

Dopo aver collocato il controllo ActiveX associato ai dati nel Form Designer, assegnare la proprietà *DataSource* alla sorgente dati che rappresenta il dataset desiderato. Dopo aver specificato un data source, si può utilizzare il DataBindings editor per collegare la proprietà relativa ai dati di un controllo a un campo del dataset.

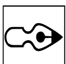
Per visualizzare il Data Bindings editor, fare clic destro sul controllo ActiveX associato ai dati in modo da visualizzare un elenco di opzioni. Oltre alle opzioni di base, apparirà anche la voce aggiuntiva Data Bindings. Selezionare l'elemento per visualizzare il Data Bindings editor che elenca i nomi dei campi nel dataset e le proprietà collegabili del controllo ActiveX.

Per collegare un campo a una proprietà:

1. Nella finestra di dialogo Data Bindings Editor di ActiveX, selezionare un campo e un nome di proprietà.

   Field Name elenca i campi del database e Property Name elenca le proprietà del controllo ActiveX che possono essere collegate a un campo del database. Il dispID della proprietà è racchiuso tra parentesi, ad esempio Value(12).

2. Fare clic su Bind e poi su OK.



Se nella finestra di dialogo non appare alcuna proprietà, significa che il controllo ActiveX non contiene proprietà associate ai dati. Per attivare l'associazione di dati semplici per una proprietà di un controllo ActiveX, utilizzare la libreria di tipi come descritto in "Attivazione dell'associazione di dati semplici con la libreria di tipi" a pagina 43-12.

L'esempio seguente illustra passo passo l'utilizzo di un controllo ActiveX associato ai dati nel contenitore di C++Builder. Questo esempio utilizza il controllo Calendar di Microsoft che è disponibile se sul sistema è installato Microsoft Office 97.

1 Dal menu principale di Delphi, scegliere l'opzione Component | Import ActiveX Control.
2 Selezionare un controllo ActiveX associato ai dati, come Microsoft Calendar control 8.0 e modificare il nome della classe in *TCalendarAXControl*, quindi fare clic su Install.
3 Nella finestra di dialogo Install, fare clic su OK per aggiungere il controllo al package utente predefinito rendendo così il controllo disponibile sulla Palette.
4 Scegliere il comando Close All e, successivamente, File | New | Application per iniziare una nuova applicazione.
5 Dalla pagina ActiveX della Component Palette, trascinare sulla scheda un oggetto *TCalendarAXControl* che è stato appena aggiunto alla Palette.
6 Trascinare sulla scheda un oggetto *DataSource* dalla pagina Data Access e un oggetto *Table* dalla pagina BDE.
7 Selezionare l'oggetto *DataSource* e impostarne la proprietà *DataSet* a *Table1*.
8 Selezionare l'oggetto *Table* e fare quanto segue:
   - Impostare la proprietà *DatabaseName* a BCDEMOS
   - Impostare la proprietà *TableName* a EMPLOYEE.DB
   - Impostate la proprietà *Active* a **true**
9 Selezionare l'oggetto *TCalendarAXControl* e impostarne la proprietà *DataSource* a *DataSource1*.
10 Selezionare l'oggetto *TCalendarAXControl*, fare clic destro e scegliere il comando Data Bindings per richiamare il Data Bindings Editor del controllo ActiveX.

   Field Name elenca tutti i campi nel database attivo. Property Name elenca quelle proprietà del Controllo ActiveX che sono collegabili ad un campo del database. Il dispID della proprietà è racchiuso tra parentesi.
11 Selezionare il campo *HireDate* e il nome della proprietà *Value*, scegliere *Bind*, quindi fare clic su OK.

   Il nome di campo e la proprietà risultano ora collegati.
12 Dalla pagina Data Controls, trascinare sulla scheda un oggetto *DBGrid* e impostarne la proprietà *DataSource* a *DataSource1*.
13 Dalla pagina Data Controls, trascinare sulla scheda un oggetto *DBNavigator* e impostarne la proprietà *DataSource* a *DataSource1*.
14 Eseguire l'applicazione.
15 Collaudare l'applicazione come segue:

Col campo *HireDate* visualizzato nell'oggetto *DBGrid*, spostarsi nel database utilizzando l'oggetto Navigator. Le date nel controllo ActiveX dovrebbero modificarsi man mano che ci si sposta nel database.

Esempio: stampa di un documento con Microsoft Word

La seguenti fasi mostrano come creare un controller Automation che stampa un documento usando Microsoft Word 8 da Office 97.

Prima di iniziare: Creare un nuovo progetto costituito da una scheda, da un pulsante e da una finestra di dialogo per l'apertura di file (*TOpenDialog*). Questi controlli costituiscono il controller Automation.

Fase 1: preparazione di C++Builder per l'esempio

Per comodità, C++Builder include nella Component Palette diversi server molto comuni, come Word, Excel e PowerPoint. Per dimostrare come si importa un server si utilizzerà Word.Poiché è già incluso nella Component palette, la prima cosa da fare sarà quella di rimuovere il package che contiene Word in modo che sia possibile installarlo nuovamente nella tavolozza. La Fase 4 descrive come riportare la Component palette allo stato originario.

Per rimuovere Word dalla Component Palette:

1. Selezionare il comando Component | Install packages.
2. Fare clic su Borland C++Builder COM Server Components Sample Package e scegliere Remove.

   La pagina Servers della Component palette non conterrà più nessuno dei server forniti con C++Builder. (Se non sono stati importati altri server, anche la pagina Servers sarà rimossa.)

Fase 2: importazione della libreria di tipi di Word

Per importare la libreria di tipi di Word:

1. Scegliere il comando Project | Import Type Library.
2. Nella finestra di dialogo Import Type Library,
   1. Selezionare Microsoft Office 8.0 Object Library.

      Se Word (Versione 8) non è incluso nell'elenco, fare clic sul pulsante Add, passare alla directory Programmi\Microsoft Office\Office, selezionare il file della libreria di tipi di Word, MSWord8.olb, fare clic su Add e selezionare finalmente Word (Versione 8) dall'elenco.
   2. Per l'opzione Palette Page, selezionare Servers.
   3. Fare clic su Install.

      Apparirà la finestra di dialogo Install. Selezionare la pagina Into New Packages e scrivere WordExample per creare un nuovo package che conterrà la libreria di tipi selezionata.

3 Selezionare la pagina Servers della Component Palette, selezionare WordApplication e collocarlo su una scheda.

4 Scrivere un gestore di evento per l'oggetto pulsante come descritto nel paragrafo successivo.

Fase 3: uso di un oggetto interfaccia VTable o dispatch per il controllo di Microsoft Word

Per controllare Microsoft Word, è possibile usare uno dei due oggetti seguenti.

Uso di un oggetto interfaccia VTable

Trascinando sulla scheda un'istanza dell'oggetto WordApplication sarà possibile accedere in modo molto semplice al controllo utilizzando un oggetto interfaccia VTable. Sarà sufficiente chiamare i metodi della classe appena creata. Nel caso di Word, questa è la classe *TWordApplication*.

1 Selezionare il pulsante, fare doppio clic sul suo gestore di eventi *OnClick* e scrivere il seguente codice di gestione dell'evento:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  if (OpenDialog1->Execute())
  {
    TVariant FileName = OpenDialog1->FileName.c_str();
    WordApplication1->Documents->Open(&FileName);

    WordApplication1->ActiveDocument->PrintOut();
  }
}
```

2 Compilare ed eseguire il programma. Facendo clic sul pulsante, Word richiederà un file da stampare.

Uso di un oggetto interfaccia dispatch

Come metodo alternativo, è possibile utilizzare un'interfaccia dispatch per il bind differito. Per utilizzare un'oggetto interfaccia dispatch, si deve creare e inizializzare l'oggetto Application utilizzando la classe wrapper dispatch _ApplicationDisp nel modo seguente. Si noti che i metodi della dispinterface sono "documentati" nel sorgente come interfacce VTable restituite, ma, in effetti, è necessario eseguirne una conversione di tipo trasformandole in interfacce dispatch.

1 Selezionare il pulsante, fare doppio clic sul suo gestore di eventi *OnClick* e scrivere il seguente codice di gestione dell'evento:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
if (OpenDialog1->Execute())
  {
    TVariant FileName = OpenDialog1->FileName.c_str();

    _ApplicationDisp MyWord;
    MyWord.Bind(CLSID_WordApplication);
```

```
    DocumentsDisp MyDocs = MyWord->Documents;
    MyDocs->Open(&FileName);

    _DocumentDisp MyActiveDoc = MyWord->ActiveDocument;
    MyActiveDoc->PrintOut();
    MyWord->Quit();
}
```

2 Compilare ed eseguire il programma. Facendo clic sul pulsante, Word richiederà un file da stampare.

Fase 4: ripristino dello stato originario

Dopo aver completato questa esercitazione, è consigliabile ripristinare lo stato originario di C++Builder.

1 Cancellare gli oggetti sulla pagina Servers:
   - Scegliere il comando Component|Install Packages.
   - Selezionare nell'elenco il package WordExample e fare clic per rimuoverlo.
   - Fare clic su Yes per confermare l'operazione.
   - Uscire dalla finestra di dialogo Install Packages facendo clic su OK.

2 Ripristinare il Borland C++Builder COM Server Components Sample Package:
   - Scegliere il comando Component|Install Packages.
   - Fare clic sul pulsante Add.
   - Nella finestra di dialogo attivata, scegliere bcb97axserver60.bpl per i componenti Office 97, o bcb2kaxserver60.bpl per i componenti Office 2000.
   - Uscire dalla finestra di dialogo Install Packages facendo clic su OK.

Scrittura del codice del client in base alle definizioni della libreria di tipi

Sebbene sia necessario utilizzare un wrapper di componenti per ospitare un controllo ActiveX, è possibile scrivere un Automation controller utilizzando solamente le definizioni della libreria di tipi che appaiono nella unit *TypeLibName*_TLB. Questo procedimento è un po' più complesso se paragonato a quello che affida tutte le operazioni al componente, specialmente se c'è la necessità di rispondere agli eventi.

Connessione a un server

Prima di poter pilotare un Automation server mediante un'applicazione controller, è necessario ottenere un riferimento ad una delle interfacce che esso supporta. Solitamente si effettua una connessione a un server mediante la sua interfaccia principale. Per esempio, ci si connettete a Microsoft Word tramite il componente WordApplication.

Se l'interfaccia principale è un'interfaccia duale, è possibile utilizzare l'oggetto creatore nel file *TypeLibName*_TLB.h. Le classi creatore hanno lo stesso nome della

CoClass, col in più il prefisso "Co". È possibile effettuare una connessione a un server sulla stessa macchina chiamando il metodo *Create*, oppure a un server su una macchina remota utilizzando il metodo *CreateRemote*. Poiché *Create* e *CreateRemote* sono metodi statici, per poterli chiamare non è necessaria un'istanza della classe creatore.

```
pInterface = CoServerClassName.Create();
pInterface = CoServerClassName.CreateRemote("Machine1");
```

Create e *CreateRemote* restituiscono il wrapper della classe per l'interfaccia predefinita della CoClass (definita in *TypeLibName*_TLB.h utilizzando il modello TComInterface.

Se l'interfaccia predefinita è un'interfaccia dispatch, non è stata generata alcuna classe Creator per la CoClass. Occorre invece creare un'istanza della classe del wrapper generata automaticamente per l'interfaccia predefinita. Questa classe è definita utilizzando il modello *TAutoDriver*, ed il suo nome assume il formato InterfaceNameDisp. In seguito, chiamate il metodo *Bind* passandogli il GUID della CoClass (all'inizio della unit _TLB viene definita una costante per questo GUID).

Controllo di un server Automation usando un'interfaccia duale

Dopo avere utilizzato la classe creatore generata automaticamente per connettersi al server, si chiamano i metodi dell'oggetto wrapper dell'interfaccia, utilizzando l'operatore "->". Ad esempio:

```
TComApplication AppPtr = CoWordApplication_.Create();
AppPtr->DoSomething;
```

Il wrapper dell'interfaccia e classe creatore sono definite nella unit *TypeLibName*_TLB, generata automaticamente al momento dell'importazione della libreria di tipi. Uno dei vantaggi derivanti dall'utilizzo di questa classe wrapper per le interfacce è che libera automaticamente l'interfaccia associata al momento della sua distruzione.

Per informazioni sulle interfacce duali, consultare "Interfacce duali" a pagina 41-13.

Controllo di un server Automation usando un'interfaccia dispatch

Di solito, per controllare un Automation server si utilizzerà l'interfaccia duale, come descritto in precedenza. Tuttavia, potrebbe sorgere la necessità di controllare un Automation server mediante un'interfaccia dispatch perché non è disponibile un'interfaccia duale.

Per chiamare i metodi di un'interfaccia dispatch:

1. Connettersi al server utilizzando il metodo *Bind* della classe wrapper dell'interfaccia dispatch. Per maggiori informazioni sulla connessione al server, consultare "Connessione a un server" a pagina 40-13.
2. Controllare l'Automation server chiamando i metodi dell'oggetto wrapper dell'interfaccia dispatch.

La classe wrapper rende evidenti le proprietà e i metodi dell'interfaccia dispatch sotto forma di sue proprietà e metodi. Inoltre, poiché deriva da *TAutoDriver*, è possibile utilizzare il meccanismo *IDispatch* per chiamare le proprietà e i metodi dell'oggetto server. Prima di poter accedere a proprietà o metodi, è necessario

otteneme l'ID di dispatch. Per fare ciò, utilizzare il metodo *GetIDsOfNames*, passandogli un nome e un riferimento a una variabile DISPID che riceverà l'ID di dispatch per quel nome. Una volta ottenuto l'ID di dispatch, è possibile utilizzarlo per chiamare *OlePropertyGet*, *OlePropertyPut* o *OleFunction* per accedere alle proprietà e ai metodi degli oggetti server.

Un altro modo per utilizzare le interfacce dispatch è quello di assegnarle a un *Variant*. Questo metodo è richiesto da alcuni oggetti VCL per quelle proprietà o parametri il cui valori sono interfacce. Il tipo *Variant* include un supporto interno per chiamare le interfacce dispatch, tramite i suoi metodi *OlePropertyGet*, *OlePropertySet*, *OleFunction* e *OleProcedure*, che corrispondono ai metodi su una classe wrapper di un'interfaccia dispatch. L'utilizzo di un *Variant* può essere un po' più lento rispetto all'utilizzo della classe wrapper di un'interfaccia dispatch, in quanto i metodi *Variant* cercano dinamicamente l'ID di dispatch ogni volta che li si chiama.Comunque, hanno il vantaggio che non è necessario importare una libreria di tipi per utilizzarli.



Occorre prestare attenzione quando si assegna un'interfaccia a un *Variant* invece che a un wrapper di interfaccia dispatch. Mentre il wrapper gestisce automaticamente le chiamate a *AddRef* e a *Release*, il *Variant* non lo fa. Così, per esempio, se si assegna a un wrapper di interfaccia un *Variant* il cui valore è un'interfaccia, il wrapper di interfaccia non chiama *AddRef*, ma il suo distruttore chiama *Release*. A causa di ciò, non si consiglia l'utilizzo di *Variant* se non nel caso in cui appaiano già come proprietà di un oggetto VCL.

Per maggiori informazioni sulle interfacce dispatch, consultare "Interfacce di Automation" a pagina 41-13.

Gestione degli eventi in un controller Automation

Quando si genera un wrapper di componenti per un oggetto di cui si importa la libreria di tipi, è possibile rispondere agli eventi utilizzando semplicemente gli eventi aggiunti al componente generato. Se non si utilizza un wrapper di componenti (o se il server utilizza eventi COM+), è necessario scrivere personalmente un convogliatore di eventi.

Gestione da programma degli eventi di Automation

Prima di poter gestire gli eventi, è necessario definire un convogliatore di eventi. Questa è una classe che implementa l'interfaccia dispatch degli eventi, definita nella libreria di tipi del server.

Il convogliatore di eventi è un discendente di *TEventDispatcher*, che è una classe basata su un modello che richiede due parametri, la classe del convogliatore di eventi e il GUID dell'interfaccia dell'evento gestita dal convogliatore di eventi:

```
class MyEventSinkClass: TEventDispatcher<MyEventSinkClass, DIID_TheServerEvents>
{
...// declare the methods of DIID_TheServerEvents here
}
```

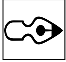
Una volta ottenuta un'istanza della classe convogliatore di eventi, chiamarne il metodo *ConnectEvents* per rendere nota al server l'esistenza del convogliatore di eventi. Questo metodo utilizza le interfacce *IConnectionPointContainer* e *IConnectionPoint* del server per registrare l'oggetto come un convogliatore di eventi.

A questo punto l'oggetto riceve le chiamate dal server non appena si verifica un evento:

```
pInterface = CoServerClassName.CreateRemote("Machine1");
MyEventSinkClass ES;
ES.ConnectEvents(pInterface);
```

Prima di poter liberare il convogliatore di eventi si deve terminare la connessione. Per fare ciò, chiamare il metodo *DisconnectEvents* del convogliatore di eventi:

```
ES.DisconnectEvents(pInterface);
```



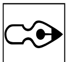
Occorre essere certi che il server abbia rilasciato la sua connessione al convogliatore di eventi prima di poterlo liberare. Poiché non si sa come il server risponde alla notifica di disconnessione iniziata da *DisconnectEvents*, ciò potrebbe trasformarsi in una specie di gara se si libera il convogliatore di eventi immediatamente dopo la chiamata. *TEventDispatcher* evita automaticamente questa situazione mantenendo un proprio contatore dei riferimenti che non viene decrementato finché il server non rilascia l'interfaccia del convogliatore di eventi.

Gestione di eventi COM+

In ambiente COM+, i server utilizzano uno speciale oggetto helper per generare eventi invece di un set di interfacce speciali (*IConnectionPointContainer* e *IConnectionPoint*). A causa di ciò, non è possibile utilizzare un convogliatore di eventi che discende da *TEventDispatcher*. *TEventDispatcher* è progettato per lavorare con quelle interfacce e non con oggetti evento di COM+.

Invece di definire un convogliatore di eventi, l'applicazione client definisce un oggetto subscriber. Gli oggetti subscriber obietta, come i convogliatori di eventi, forniscono l'implementazione dell'interfaccia dell'evento. Differiscono dai convogliatori di eventi per il fatto che si "abbonano" ad un particolare oggetto evento piuttosto che connettersi a un punto di connessione di un server.

Per definire un oggetto subscriber, utilizzare il COM Object wizard, selezionando l'interfaccia dell'oggetto evento come quella che si desidera implementare. Il wizard genera una unit di implementazione con l'abbozzo dei metodi che si dovranno completare per creare i gestori di evento. Per ulteriori informazioni sull'utilizzo del COM Object wizard per implementare un'interfaccia esistente, consultare "Utilizzo di COM object wizard" a pagina 41-3.



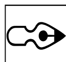
Potrebbe essere necessario aggiungere l'interfaccia dell'oggetto evento al file di registro utilizzando il wizard, nel caso non appaia nell'elenco delle interfacce che è possibile implementare.

Una volta creato l'oggetto subscriber, si deve effettuare la sottoscrizione all'interfaccia dell'oggetto evento o ai singoli metodi (gli eventi) su quella interfaccia. Esistono tre tipi di sottoscrizioni fra cui scegliere:

- **Sottoscrizioni transitorie**. Analogamente ai convogliatori di evento tradizionali, le sottoscrizioni transitorie sono legate alla durata di un'istanza dell'oggetto. Quando l'oggetto subscriber viene liberato, la sottoscrizione termina e COM+ non gli spedisce più alcun evento.

- **Sottoscrizioni permanenti**. Queste sono legate alla classe dell'oggetto invece che a un'istanza specifica dell'oggetto. Quando l'evento si verifica, COM individua o avvia un'istanza dell'oggetto subscriber e ne chiama il gestore di evento. Gli oggetti in-process (DLL) utilizzano questo tipo di sottoscrizione.
- **Sottoscrizioni per-utente.** Queste sottoscrizioni rappresentano una versione più sicura delle sottoscrizioni transitorie. Sia l'oggetto subscriber sia l'oggetto server che scatena gli eventi, devono essere in esecuzione nello stesso account di utente sulla stessa macchina.

Per effettuare la sottoscrizione a un oggetto evento, utilizzare la funzione globale *RegisterComPlusEventSubscription*.



Gli oggetti che effettuano una sottoscrizione a eventi COM+ devono essere installati in un'applicazione di COM+.

Creazione di client per server che non hanno una libreria di tipi

Alcune vecchie tecnologie COM, come ad esempio OLE (object linking and embedding), non forniscono informazioni di tipo in una libreria di tipi. Esse si appoggiano invece ad una serie standard di interfacce predefinite. Per scrivere client in grado di ospitare tali oggetti, è possibile utilizzare il componente *TOleContainer*. Questo componente è presente sulla pagina System della Component palette.

TOleContainer si comporta come sito ospitante per un oggetto Ole2. Implementa l'interfaccia *IOleClientSite* e, facoltativamente, *IOleDocumentSite*. La comunicazione viene gestita utilizzando verbi OLE.

Per utilizzare *TOleContainer*:

1. Collocare un componente *TOleContainer* sulla scheda.
2. Impostare a **true** la proprietà *AllowActiveDoc* se si desidera ospitare un *Active* document.
3. Impostare la proprietà *AllowInPlace* per indicare se l'oggetto ospitato deve apparire nel *TOleContainer* o in una finestra separata.
4. Scrivete i gestori di evento per rispondere all'attivazione, alla disattivazione, allo spostamento o al ridimensionamento dell'oggetto.
5. Per eseguire il bind dell'oggetto *TOleContainer* in fase di progetto, fare clic destro e scegliere il comando Insert Object. Nella finestra di dialogo Insert Object, scegliere un oggetto server da ospitare.
6. Per eseguire il bind dell'oggetto *TOleContainer* in fase esecuzione, esistono molti metodi fra cui scegliere, a seconda di come si desidera identificare l'oggetto server. Fra questi vi sono *CreateObject*, che richiede un id di programma, *CreateObjectFromFile*, che richiede il nome di un file in cui è stato salvato l'oggetto, *CreateObjectFromInfo*, che richiede un struct contenente informazioni su come creare l'oggetto, oppure *CreateLinkToFile*, che richiede il nome di un file in cui è stato salvato l'oggetto e a cui si collega invece che incorporarlo.

7 Una volta collegato l'oggetto, è possibile accedere alla sua interfaccia utilizzando la proprietà *OleObjectInterface*. Tuttavia, poiché la comunicazione con gli oggetti Ole2 era basata su verbi OLE, molto più probabilmente si trasmetteranno comandi al server utilizzando il metodo *DoVerb*.

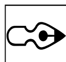
8 Quando si desidera rilasciare l'oggetto server, chiamare il metodo *DestroyObject*.

Capitolo

41

Creazione di semplici server COM

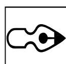
C++Builder fornisce wizard che facilitano la creazione di diversi oggetti COM. Gli oggetti COM più semplici sono server che mettono a disposizione proprietà e metodi (e possibilmente eventi) attraverso un'interfaccia predefinita che può essere chiamata dai client.



Server COM e Automation non sono disponibili per l'uso in applicazioni CLX. Questa tecnologia è da utilizzare solo in ambiente Windows e non è per multipiattaforma.

Due wizard, in particolare, facilitano il processo di creazione di semplici oggetti COM:

- COM Object wizard costruisce un oggetto COM leggero la cui interfaccia predefinita deriva da *IUnknown* o che implementa un'interfaccia già registrata nel sistema. Questa procedura guidata fornisce il livello di flessibilità massimo nei tipi di oggetti COM che è possibile creare.
- Automation Object wizard crea un semplice oggetto Automation la cui interfaccia predefinita deriva da *IDispatch*. *IDispatch* introduce un meccanismo standard di marshaling e supporta il binding differito delle chiamate di interfaccia.



COM definisce molte interfacce e meccanismi standard per la gestione di situazioni specifiche. I wizard di C++Builder automatizzano la maggior parte delle attività più comuni. Tuttavia, alcune attività, come il marshaling personalizzato, non sono supportate da alcuna procedura guidata di C++Builder. Per informazioni su questa e altre tecnologie non esplicitamente supportate da C++Builder, fare riferimento alla documentazione di Microsoft Developer's Network (MSDN). Anche il sito Web di Microsoft fornisce informazioni aggiornate sul supporto COM.

Panoramica sulla creazione di un oggetto COM

Automation object wizard, per la creazione di un nuovo server di automazione, e COM object wizard, per la creazione di altri tipi di oggetto COM, sono caratterizzati dallo svolgimento dello stesso processo. Di seguito sono descritte le varie fasi:

1. Progettazione dell'oggetto COM.
2. Uso di COM Object wizard o Automation Object wizard per creare l'oggetto server.
3. Specifica delle opzioni nella pagina ATL della finestra di dialogo delle opzioni di progetto per indicare in che modo l'oggetto COM chiamerà l'applicazione che ospita l'oggetto e quale tipo di supporto di debugging si desidera.
4. Definizione dell'interfaccia che l'oggetto mostra ai client.
5. Registrazione dell'oggetto.
6. Collaudo e debug dell'applicazione.

Progettazione di un oggetto COM

Quando si progetta l'oggetto COM, bisogna decidere quale interfaccia si intende implementare. È possibile scrivere un oggetto COM per implementare un'interfaccia che è già stata definita, oppure è possibile definire una nuova interfaccia che l'oggetto deve implementare. Inoltre, è possibile fare in modo che l'oggetto supporti più di un'interfaccia. Per informazioni sulle interfacce COM standard che è possibile supportare, consultare la documentazione di MSDN.

- Per creare un oggetto COM che implementa un'interfaccia esistente, utilizzare COM Object wizard.
- Per creare un oggetto COM che implementa una nuova interfaccia da definire, utilizzare COM Object wizard o Automation Object wizard. COM object wizard può generare una nuova interfaccia predefinita che deriva da *IUnknown*, e Automation object wizard fornisce all'oggetto un'interfaccia predefinita che deriva da *IDispatch*. Indipendentemente dalla procedura guidata scelta è sempre possibile utilizzare successivamente il Type Library editor per modificare l'interfaccia genitore o quella predefinita generata dalla procedura guidata.

Oltre a decidere quali interfacce supportare, si deve stabilire l'oggetto COM se è destinato ad essere un server in-process un server out-of-process o un server remoto. Per tutti questi tipi di server, comunque, il ricorso ad una libreria di tipi COM fa sì che i dati vengano disposti automaticamente. Altrimenti, si deve prendere in considerazione come disporre i dati nei server esterni al processo. Per informazioni sugli altri tipi di server, consultare il paragrafo, "Server in-process, out-of-process e remoti", a pagina 38-6.

Utilizzo di COM object wizard

Il COM object wizard esegue le seguenti operazioni:

- Crea una nuova unit.
- Definisce una nuova classe che deriva dalle classi ATL CComObjectRootEx e CComCoClass. Per maggiori informazioni sulle classi di base, vedere “Codice generato dai wizard” a pagina 38-23.
- Aggiunge una libreria di tipi al progetto e vi aggiunge l’oggetto e la relativa interfaccia.

Prima di creare un oggetto COM, bisogna creare o aprire il progetto per l’applicazione che contenga la funzionalità che si intende implementare. Il progetto può essere un’applicazione o una libreria ActiveX, in base alle proprie necessità.

Per richiamare COM object wizard:

1. Scegliere File | New | Other per aprire la finestra di dialogo New Items.
2. Selezionate la pagina identificata con ActiveX.
3. Fare doppio clic sull’icona COM object.

Nel wizard si deve specificare quanto segue:

- **Nome della CoClass:** questo è il nome dell’oggetto così come viene visualizzato ai client. La classe creata per implementare l’oggetto ha questo nome preceduto da una lettera “T”. Se si sceglie di non implementare un’interfaccia esistente, il wizard assegna alla CoClass un’interfaccia predefinita che ha questo nome preceduto da una lettera ‘I’.
- **Interfaccia da implementare:** come impostazione predefinita, il wizard assegna all’oggetto un’interfaccia predefinita che deriva da *IUnknown*. Dopo l’uscita dal wizard, è possibile utilizzare il Type Libary editor per aggiungere a questa interfaccia proprietà e metodi. Tuttavia, è anche possibile selezionare un’interfaccia predefinita da implementare nell’oggetto. Fare clic sul pulsante List in COM object wizard per richiamare l’Interface Selection wizard, dove è possibile selezionare qualsiasi interfaccia duale o personalizzata definita in una libreria di tipi registrata nel sistema. L’interfaccia che si seleziona diventa quella predefinita per la nuova CoClass. Il wizard aggiunge tutti i metodi presenti in questa interfaccia alla classe di implementazione generata, quindi è necessario solamente compilare il corpo dei metodi nella unit di implementazione. Notare che se si seleziona un’interfaccia esistente, questa non viene aggiunta alla libreria di tipi del progetto. Questo significa che quando si sviluppa l’oggetto è necessario anche impiegare la libreria di tipi che definisce l’interfaccia.
- **Modello di threading:** tipicamente, le richieste del client all’oggetto entrano in thread di esecuzione diversi. È possibile specificare il modo in cui COM serializza questi thread quando chiama l’oggetto. La scelta del modello di threading determina come vengono registrati gli oggetti. Il programmatore è responsabile di fornire ogni supporto di threading necessario per il modello scelto. Per informazioni sulle differenti possibilità, vedere “Scelta di una modello di

threadingl" a pagina 41-5. Per informazioni su come fornire supporto thread all'applicazione, vedere il Capitolo 11, "Scrittura di applicazioni multi-thread".

- **Supporto degli eventi:** è necessario indicare se si desidera che l'oggetto generi eventi a cui i client possono rispondere. Il wizard può fornire supporto per le interfacce necessarie per la generazione di eventi e lo smistamento di chiamate ai gestori di eventi client. Per informazioni sul funzionamento degli eventi e su ciò che è necessario per la loro implementazione, vedere "Esposizione di eventi ai client" a pagina 41-11.
- **Oleautomation:** se l'intenzione è quella di limitarsi all'uso di tipi compatibili con Automation, è possibile lasciare che COM gestisca automaticamente il marshaling quando non si genera un server in-process. Contrassegnando l'interfaccia dell'oggetto come OleAutomation nella libreria di tipi, si abilita COM a configurare automaticamente i proxy e gli stub e a gestire il passaggio dei parametri attraverso i confini del processo. Per maggiori informazioni su questo processo, vedere "Il meccanismo di smistamento" a pagina 38-8. Se si sta generando una nuova interfaccia è possibile specificare solo se essa è compatibile con Automation. Se si sta generando una nuova interfaccia è possibile specificare solo se essa è compatibile con Automation. Se si seleziona un'interfaccia esistente, i suoi attributi sono già specificati nella relativa libreria di tipi. Se l'interfaccia dell'oggetto non è contrassegnata come OleAutomation, è necessario creare un server in-process oppure scrivere del proprio codice per il marshaling.
- **Implement Ancestor Interfaces:** Scegliere questa opzione se si desidera che il wizard generi automaticamente le routine prototipo per le interfacce ereditate. Vi sono tre interfacce ereditate che non saranno mai implementate dal wizard: *IUnknown*, *IDispatch* e *IAppServer*. *IUnknown* e *IDispatch* non vengono implementate in quanto, per queste due interfacce, ATL fornisce la propria implementazione. *IAppServer* non viene implementato in quanto viene implementato automaticamente quando si opera con i dataset client e con i provider di dataset.

È possibile aggiungere opzionalmente una descrizione dell'oggetto COM. Questa descrizione viene visualizzata nella libreria di tipi per l'oggetto.

Uso di Automation object wizard

The Automation object wizard svolge le seguenti operazioni:

- Crea una nuova unit.
- Definisce una nuova classe che deriva dalle classi ATL *CComObjectRootEx* e *CComCoClass*. Per maggiori informazioni sulle classi di base, vedere "Codice generato dai wizard" a pagina 38-23.
- Aggiunge una libreria di tipi al progetto e vi aggiunge l'oggetto e la relativa interfaccia.

Prima di creare un oggetto Automation, creare o aprire il progetto per un'applicazione contenente le funzionalità da mettere a disposizione. Il progetto può essere un'applicazione o una libreria ActiveX, in base alle proprie necessità.

Per visualizzare Automation wizard:

1 Scegliere il comando File|New|Other.

2 Selezionate la pagina identificata con ActiveX.

3 Fare doppio clic sull'icona Automation Object.

Nella finestra di dialogo del wizard, specificare le seguenti informazioni:

- **Nome della CoClass:** questo è il nome dell'oggetto così come viene visualizzato ai client. L'interfaccia predefinita dell'oggetto viene creata con un nome basato su quello di questa CoClass preceduto da una lettera 'I', e la classe creata per implementare l'oggetto ha questo nome preceduto da una lettera 'T.
- **Modello di threading**: tipicamente, le richieste del client all'oggetto entrano in thread di esecuzione diversi. È possibile specificare il modo in cui COM serializza questi thread quando chiama l'oggetto. La scelta del modello di threading determina come vengono registrati gli oggetti. Il programmatore è responsabile di fornire ogni supporto di threading necessario per il modello scelto. Per informazioni sulle differenti possibilità, vedere "Scelta di una modello di threadingl" a pagina 41-5. Per informazioni su come fornire supporto thread all'applicazione, vedere il Capitolo 11, "Scrittura di applicazioni multi-thread".
- **Supporto degli eventi**: è necessario indicare se si desidera che l'oggetto generi eventi a cui i client possono rispondere. Il wizard può fornire supporto per le interfacce necessarie per la generazione di eventi e lo smistamento di chiamate ai gestori di eventi client. Per informazioni sul funzionamento degli eventi e su ciò che è necessario per la loro implementazione, vedere "Esposizione di eventi ai client" a pagina 41-11.

È possibile aggiungere opzionalmente una descrizione dell'oggetto COM. Questa descrizione viene visualizzata nella libreria di tipi per l'oggetto.

L'oggetto Automation implementa un'**interfaccia duale**, che supporta il binding anticipato (fase di compilazione) attraverso VTable e il binding differito (fase di esecuzione) attraverso l'interfaccia *IDispatch*. Per maggiori informazioni, vedere "Interfacce duali" a pagina 41-13.

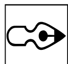
Scelta di una modello di threadingl

Quando si crea un oggetto usando un wizard, si seleziona un modello di threading che l'oggetto può supportare. Aggiungendo il supporto dei thread all'oggetto COM, se ne migliorano le prestazioni, perché i client multipli possono accedere all'applicazione contemporaneamente.

La Tabella 41.1 elenca i diversi modelli di threading che è possibile specificare.

Tabella 41.1 Modelli di threading per gli oggetti COM

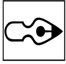
Modello di threading	Descrizione	Pro e contro dell'implementazione
Singolo	Il server non fornisce alcun supporto dei thread. COM serializza le richieste dei client in modo che l'applicazione ne riceva una alla volta.	I client vengono gestiti singolarmente, e pertanto non è necessario il supporto di alcun thread. Non ci sono vantaggi a livello di prestazioni.
Apartment (or Single-threaded apartment)	COM assicura che solo un thread client alla volta possa chiamare l'oggetto. Tutte le chiamate dei client utilizzano il thread in cui è stato creato l'oggetto.	Gli oggetti possono accedere in modo sicuro ai relativi dati di istanza, ma i dati globali devono essere protetti utilizzando le sezioni critiche o qualche altra forma di serializzazione. Le variabili locali del thread sono affidabili per più chiamate. C'è qualche vantaggio a livello di prestazioni.
Free (definito anche multi-threaded apartment)	Gli oggetti possono ricevere chiamate da qualsiasi numero di thread alla volta.	Gli oggetti devono proteggere tutte le istanze e i dati globali usando le sezioni critiche o qualche altra forma di serializzazione. Le variabili locali del thread non sono affidabili per più chiamate.
Both	L'opzione è analoga al modello Free-threaded con la differenza che le chiamate in uscita (ad esempio, le callback) vengono sempre eseguite nello stesso thread.	Massime prestazioni e flessibilità. Non richiede che l'applicazione fornisca il supporto dei thread per i parametri forniti alle chiamate in uscita.
Neutral	Più client possono chiamare contemporaneamente l'oggetto su thread differenti, ma COM assicura che non vi siano due chiamate in conflitto.	È necessario proteggersi dai conflitti di thread che coinvolgono dati globali e da qualsiasi dato di istanza a cui accedono più metodi. Questo modello non va usato con oggetti che hanno un'interfaccia utente (controlli visuali). Questo modello è disponibile solo in COM+. Sotto COM, viene mappato sul modello Apartment.



Le variabili locali (tranne quelle nelle callback) sono sempre sicure, indipendentemente dal modello di threading. La ragione è che le variabili locali sono memorizzate nello stack e ciascun thread ha il proprio stack. Le variabili locali possono non essere sicure nelle callback quando si utilizza il threading free.

Il modello di threading scelto nel wizard determina come viene registrato l'oggetto nel Registry di sistema. Bisogna accertarsi che l'implementazione dell'oggetto aderisca al modello di threading che è stato scelto. Per informazioni generali sulla

scrittura di codice thread sicuro, consultare il Capitolo 11, “Scrittura di applicazioni multi-thread”.



Affinché COM chiami l’oggetto in base al modello di threading specificato, è necessario che venga inizializzato il supporto per quel tipo di modello. È possibile specificare come viene inizializzato COM tramite la pagina ATL della finestra di dialogo di opzioni del progetto.

Per server in-process, l’impostazione del modello di threading nel wizard definisce la chiave del modello di threading sulla voce di registro CLSID.

I server a processo esterno vengono registrati come EXE, e COM deve essere inizializzato per il modello di threading più alto necessario. Ad esempio, se un EXE include un oggetto con threading free, deve essere inizializzato per tale modello, e ciò significa che può fornire il supporto previsto per qualsiasi oggetto con threading free o apartment contenuto nell’EXE. Per specificare in che modo COM viene inizializzato, utilizzare la pagina ATL della finestra di dialogo di opzioni del progetto.

Scrittura di un oggetto che supporta il modello free threading

Utilizzare il modello di threading free (o entrambi) invece di quello apartment quando è necessario che più di un thread possa accedere all’oggetto. Un esempio comune è un’applicazione client connessa ad un oggetto su una macchina remota. Quando il client remoto chiama un metodo di quell’oggetto, il server riceve la chiamata su un thread da un insieme di thread della macchina server. Questo thread ricevente effettua la chiamata a livello locale all’oggetto vero e proprio; e, dal momento che l’oggetto supporta il modello di threading free, il thread può effettuare una chiamata diretta nell’oggetto.

Se, invece, l’oggetto supportasse il modello di threading apartment, la chiamata dovrebbe essere trasferita al thread sul quale l’oggetto è stato creato, e il risultato sarebbe ritrasferito al thread ricevente prima di ritornare al client. Questo approccio richiede un ulteriore smistamento.

Per supportare il threading free, occorrono considerare il modo in cui è possibile accedere ai dati dell’istanza per *ciascun* metodo. Se il metodo è la scrittura nei dati dell’istanza, per proteggere i dati dell’istanza si devono utilizzare le sezioni critiche o qualche altra forma di serializzazione. Probabilmente, l’overhead delle chiamate critiche di serializzazione è minore dell’esecuzione del codice di smistamento di COM.

Si noti che, se i dati dell’istanza sono a sola lettura, la serializzazione non è necessaria.

I server in-process free-threaded possono migliorare le prestazioni agendo come oggetto esterno un’aggregazione con il gestore del marshaling free-threaded. Quest’ultimo fornisce una scorciatoia per la gestione dei thread standard di COM quando viene chiamata una DLL free threaded da parte di un host (client) che non è free threaded.

Per eseguire l’aggregazione con il gestore di marshaling free threaded, è necessario

- Chiamare *CoCreateFreeThreadedMarshaler*, passando l'interfaccia *IUnknown* dell'oggetto per il gestore di marshaling free threaded da utilizzare:

  ```
  CoCreateFreeThreadedMarshaler(static_cast<IUnknown *>(this), &FMarshaler);
  ```

  Questa riga assegna all'interfaccia per il gestore di marshaling free threaded un numero di classe, *FMarshaler*.
- Tramite il Type Libary editor, aggiungere l'interfaccia *IMarshal* al set di interfacce implementate da CoClass.
- Nel metodo *QueryInterface* dell'oggetto, delegare le chiamate per IDD_IMarshal al gestore di marshaling free threaded (memorizzato come *FMarshaler* sopra).



Il gestore di marshaling free threaded viola le normali regole del marshaling COM per fornire una maggiore efficienza. Lo si deve utilizzare con cautela. In particolare, essa va aggregato solo con oggetti free threaded in server in-process, e va istanziato solo dall'oggetto che lo utilizza (non da un altro thread).

Scrittura di un oggetto che supporta il modello apartment threading

Per implementare il modello di threading apartment (single-threaded), occorre seguire poche regole:

- Il primo thread dell'applicazione che viene creato è il thread COM principale. Questo è generalmente il thread in cui viene chiamato WinMain. Esso deve anche essere l'ultimo thread che cessa di inizializzare COM.
- Ogni thread nel modello di threading apartment deve avere un ciclo di messaggi e la coda dei messaggi deve essere controllata frequentemente.
- Quando un thread fornisce un puntatore ad un'interfaccia COM, quel puntatore può essere usato solo da quel thread.

Il modello single-threaded apartment è il livello intermedio tra il supporto completo multithreading del modello free e l'assenza del supporto del thread. Un server che si affida al modello apartment garantisce che il server avrà un accesso serializzato a tutti i propri dati globali (come il proprio contatore di oggetti). Un server che si affida al modello apartment garantisce di avere l'accesso serializzato a tutti i suoi dati globali (come il suo numero di oggetti), in quanto oggetti diversi possono tentare di accedere ai dati globali da thread diversi. Tuttavia, i dati dell'istanza dell'oggetto sono al sicuro, in quanto i metodi vengono sempre chiamati sullo stesso thread.

Generalmente, i controlli utilizzati nei browser per Web usano il modello di threading apartment, in quanto le applicazioni browser inizializzano sempre i loro thread come apartment.

Scrittura di un oggetto che supporta il modello di threading neutral

Sotto COM+, è possibile utilizzare un altro modello di threading che sta a metà strada tra il threading free e il threading apartment: il modello neutral. Come il modello di threading free, questo consente che molteplici thread accedano contemporaneamente all'oggetto. Non c'è alcun marshaling aggiuntivo per il trasferimento al thread in cui è stato creato l'oggetto. Tuttavia, viene garantito che l'oggetto non riceva chiamate in conflitto.

La scrittura di un oggetto che utilizza il modello di threading neutrale segue le stesse regole della scrittura di un oggetto a thread apartment, tranne che è necessario difendere i dati di istanza da conflitti di thread se essi possono essere richiamati da metodi differenti nell'interfaccia dell'oggetto. Qualsiasi dato di istanza a cui accede solo un singolo metodo di interfaccia è protetto automaticamente.

Specifica di opzioni ATL

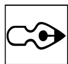
Dopo avere utilizzato un wizard per creare un oggetto COM, la finestra di dialogo Project Options mette a disposizione una pagina aggiuntiva etichettata "ATL". Questa pagina consente di impostare un numero di flag a livello di applicazione che controllano quali parametri vengono generati per chiamate ATL che inizializzano COM o registrano l'applicazione, oltre a flag che controllano se viene attivata la traccia di debug nel codice che utilizza ATL.

Utilizzare la pagina ATL per impostare le seguenti opzioni:

- **Instancing:** se l'applicazione non è un server in-process, l'installazione determina in che modo possono essere creare molte istanze degli oggetti client all'interno di un singolo spazio di processo. Se si specifica l'uso singolo, allora dopo che un client ha istanziato l'oggetto, COM rimuove l'applicazione dalla vista in modo che altri client possano eseguire le proprie istanze dell'applicazione. Se si specifica l'uso multiplo, allora più client possono creare ciascuno la propria istanza dell'oggetto, e tutte le istanze vengono eseguite nello spesso spazio di processo.
- **OLE Initialization COINIT_XXX flag:** questo flag determina in che modo COM viene inizializzato dall'applicazione. È possibile specificare il modello di threading apartment, nel qual caso ciascun oggetto viene sempre chiamato nel thread in cui è stato creato, o multi-thread, nel qual caso gli oggetti possono essere chiamati in molteplici thread. I flag COINIT determinano quali modelli di threading possono essere utilizzati dagli oggetti nell'applicazione. Se il flag COINIT è impostato su apartment-threaded, nell'applicazione è possibile avere solo oggetti a thread singolo o apartment. L'impostazione predefinita per oggetti registrati con qualsiasi altro modello di threading è apartment-threaded.
- **Debugging flags:** You can è anche possibile impostare flag per specificare che le chiamate ATL vengono registrate nel registro degli eventi quando l'applicazione viene eseguita nell'IDE. È possibile tracciare le chiamate ai metodi *IUnknown* (chiamate QueryInterface e conteggio dei riferimenti) o tutte le chiamate ATL.

Definizione dell'interfaccia di un oggetto COM

Quando si utilizza un wizard per creare un oggetto COM, la procedura genera automaticamente una libreria di tipi. La libreria di tipi fornisce alle applicazioni host un modo per scoprire ciò che l'oggetto può fare. Essa consente anche di definire l'interfaccia dell'oggetto tramite il Type Library editor. Le interfacce che si specificano nel Type Library editor definiscono quali proprietà, metodi ed eventi l'oggetto mette a disposizione dei client.



Se si seleziona un'interfaccia esistente nel wizard COM object, non è necessario aggiungere proprietà e metodi. La definizione dell'interfaccia viene importata dalla libreria di tipi in cui è stata definita. Invece, è sufficiente localizzare i metodi dell'interfaccia importata nella unit di implementazione e compilare il relativo corpo.

Aggiunta di una proprietà all'interfaccia dell'oggetto

Quando si aggiunge una proprietà all'interfaccia dell'oggetto tramite il Type Libary editor, viene aggiunto automaticamente un metodo per leggere il valore della proprietà e/o un metodo per impostare il valore della proprietà. Il Type Library editor, a sua volta, aggiunge questi metodi alla classe di implementazione, e nella unit di implementazione crea implementazioni di metodo vuote da completare.

Per aggiungere una proprietà all'interfaccia dell'oggetto:

1 Nell'editor della libreria dei tipi, selezionare l'interfaccia duale per l'oggetto.

   L'interfaccia predefinita dovrebbe avere lo stesso nome dell'oggetto, con anteposta la lettera "I". Per determinare l'interfaccia predefinita, nel Type Library editor scegliere il separatore CoClass and Implements e controllare nell'elenco delle interfacce implementate quella contrassegnata con "Default."

2 Per esporre una proprietà lettura/scrittura, fare clic sul pulsante Property nella barra degli strumenti; diversamente, fare clic sulla freccia accanto al pulsante Property nella barra degli strumenti, quindi fare clic sul tipo di proprietà da esporre.

3 Nel quadro Attributes, specificare il nome e il tipo di proprietà.

4 Nel quadro Attributes, specificare il nome e il tipo di proprietà.

   Nella unit di implementazione dell'oggetto viene inserita una definizione e delle implementazioni di partenza per i metodi di accesso della proprietà.

5 Nella unit di implementazione localizzare i metodi di accesso per la proprietà. I loro nomi hanno la forma get_NomeProprietà e NomeProprietà, e includono solo il codice per catturare eccezioni e restituire un HRESULT. Aggiungere codice (tra le istruzioni **try** e **catch**) che ottenga o imposti il valore della proprietà dell'oggetto. Questo codice può chiamare semplicemente una funzione esistente all'interno dell'applicazione, accedere a membri dati che si aggiungono alla definizione dell'oggetto, o implementare la proprietà in altri modi.

Aggiunta di un metodo all'interfaccia dell'oggetto

Quando si aggiunge un metodo all'interfaccia dell'oggetto tramite il Type Libary editor, questo può, a sua volta, aggiungere i metodi alla classe di implementazione, e nella unit di implementazione creare implementazione vuota da completare.

Per esporre un metodo attraverso l'interfaccia dell'oggetto:

1 Nel Type Library editor, selezionare l'interfaccia predefinita per l'oggetto

L'interfaccia predefinita dovrebbe avere lo stesso nome dell'oggetto, con anteposta la lettera "I". Per determinare l'interfaccia predefinita, nel Type Library editor scegliere il separatore CoClass and Implements e controllare nell'elenco delle interfacce implementate quella contrassegnata con "Default."

2 Fare clic sul pulsante Method.

3 Nel quadro Attributes, specificare il nome del metodo.

4 Nel quadro Parameters, specificare il tipo restituito dal metodo e aggiungere i parametri appropriati.

5 Nella barra degli strumenti, fare clic sul pulsante Refresh.

   Nella unit di implementazione dell'oggetto viene inserita una definizione e delle implementazioni di partenza per il metodo.

6 Nella unit di implementazione localizzare l'implementazione del nuovo metodo inserito. Il metodo è completamente vuoto. Compilare il corpo per svolgere l'operazione a cui è destinato il metodo.

Esposizione di eventi ai client

Un oggetto COMpuò generare due eventi: eventi tradizionali ed eventi COM+.

- Gli eventi COM+ richiedono la creazione di un oggetto evento separato tramite il wizard Event object e l'aggiunta di codice per chiamare l'oggetto evento dall'oggetto server. Per maggiori informazioni sulla generazione di eventi COM+, vedere "Generazione di eventi in ambiente COM+" a pagina 44-22.
- È possibile utilizzare il wizard per gestire la maggior parte del lavoro di generazione di eventi tradizionali. Questo processo è descritto di seguito.

Affinché un oggetto generi eventi, è necessario svolgere le seguenti operazioni:

1 Nel wizard, selezionare la casella, Generateevent support code.

   Il wizard crea un oggetto che include un'interfaccia Events oltre all'interfaccia utente predefinita. Il nome di questa interfaccia Events ha la forma *ICoClassnameEvents*. È un'interfaccia rivolta all'esterno (origine), cioè non si tratta di un'interfaccia che l'oggetto implementa, ma un'interfaccia che i client devono implementare e che viene chiamata dall'oggetto. Ciò può essere verificato selezionando la propria CoClass, andando alla pagina Implements, e osservando che la colonna Source nell'interfaccia Events indica true. La GUID per l'interfaccia Events viene aggiunta alla mappa Connection Points, che appare sotto la mappa dell'interfaccia nella dichiarazione dell'oggetto. Per maggiori informazioni sulla mappa Connection Points, vedere la documentazione ATL.

   Oltre all'interfaccia Events, il wizard aggiunge altre classi di base all'oggetto. Queste include un'implementazione dell'interfaccia *IConnectionPointContainer* (*IConnectionPointContainerImpl*) e una classe modello di base che gestisce

l'attivazione di eventi per tutti i client (TEvents_*CoClassName*). Il modello per quest'ultima classe è disponibile nello header della unit _TLB.

2 In Type Library editor selezionare l'interfaccia Events in uscita per l'oggetto. (L'interfaccia è quella il cui nome è nel formato *ICoClassNameEvents*)
3 Fare clic sul pulsante Method sulla barra strumenti della Type Library. Ogni metodo che si aggiunge all'interfaccia Events rappresenta un gestore di evento che il client deve implementare.
4 Nel quadro Attributes, specificare il nome del gestore di eventi, come MyEvent.
5 Nella barra degli strumenti, fare clic sul pulsante Refresh.

   A questo punto l'implementazione dell'oggetto ha tutto il necessario per accettare sink di eventi e mantenere un elenco di interfacce da chiamare quando se ne verifica uno.Per chiamare queste interfacce è possibile chiamare il metodo che attiva un evento (implementato da TEvents_*CoClassName*) per generare l'evento sui client. Per ciascun gestore di eventi, questo metodo ha un nome nella forma Fire_*EventHandlerName*.

6 Ogni volta che è necessario attivare l'evento in modo che i client siano informati della sua occorrenza, chiamare il metodo che smista l'evento a tutti i sink di eventi:

```
if (EventOccurs) Fire_MyEvent; // Call method you created to fire events.
```

Gestione degli eventi nell'oggetto Automation

Per supportare eventi COM tradizionali un server deve fornire la definizione di un' interfaccia di origine che viene implementata da un client. Questa interfaccia di origine include tutti i gestori di evento che il client deve implementare per rispondere agli eventi del server.

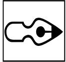
Quando un client ha implementato l'interfaccia di eventi di origine, registra il suo interesse a ricevere la notifica dell'evento inviando una query all'interfaccia *IConnectionPointContainer* del server. L'interfaccia *IConnectionPointContainer* restituisce l'interfaccia *IConnectionPoint* del server, che il client utilizza per passare al server un puntatore alla propria implementazione dei gestori di evento (nota come sink).

Il server mantiene un elenco di tutti i sink client e chiama i loro metodi quando si verifica un evento, come è stato descritto sopra.

Interfacce di Automation

Per impostazione predefinita, il wizard Automation Object implementa l'interfaccia duale, il che significa che entrambi gli oggetti Automation supportano

- Il binding differito in esecuzione, che avviene tramite l'interfaccia *IDispatch*. Questa viene implementata come un'interfaccia di dispatch o **dispinterface**.
- Il binding anticipato in fase di compilazione, che avviene tramite una chiamata diretta a una delle funzioni membro presenti nella tabella virtuale delle funzioni dell'oggetto (VTable). Questo modo viene anche detto **interfaccia custom**.



Qualsiasi interfaccia generata dal wizard COM object che non deriva da *IDispatch* supporta solo chiamate VTable.

Interfacce duali

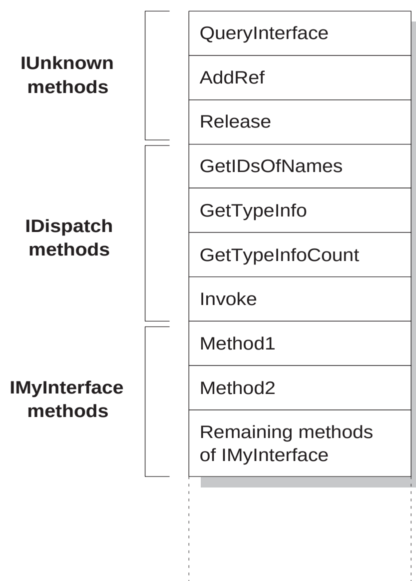
Una interfaccia duale è allo stesso tempo un'interfaccia custom e una dispinterface. Viene implementata come un'interfaccia VTable di COM derivata da *IDispatch*. Per quei controller che possono accedere all'oggetto solo in fase di esecuzione, è disponibile la dispinterface. Per gli oggetti che possono avvalersi del binding in fase di compilazione, viene utilizzata invece la più efficiente interfaccia VTable.

Le interfacce duali presentano i seguenti vantaggi combinati delle interfacce VTable e delle dispinterface:

- Per i controller Automation che non sono in grado di ottenere informazioni di tipo, la dispinterface consente di accedere all'oggetto in esecuzione.
- Per i server in-process, tramite le interfacce VTable si ottiene il beneficio di un'alta velocità di accesso.
- Per i server out-of-process, COM provvede ad eseguire il marshaling dei dati sia per le interfacce VTable sia per le dispinterface. COM possiede un'implementazione proxy/stub generica in grado di eseguire il marshaling delle interfacce in base alle informazioni di tipo contenute in una libreria di tipi. Per maggiori informazioni sul marshaling, consultare, "Marshaling dei dati", a pagina 41-15.

Il seguente diagramma illustra l'interfaccia *IMyInterfacein* un oggetto che supporta un'interfaccia duale di nome *IMyInterface*. Le prime tre voci della VTable di un'interfaccia duale fanno riferimento all'interfaccia *IUnknown*, la quattro voci successive fanno riferimento all'interfaccia *IDispatch* e la voci rimanenti sono voci COM per l'accesso diretto ai membri dell'interfaccia custom.

Figura 41.1 VTable di un'interfaccia duale



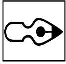
Interfacce di dispatch

I controller Automation sono client che usano l'interfaccia *IDispatch* di COM per accedere agli oggetti server di COM. Il controller deve per prima cosa creare l'oggetto, quindi richiedere all'interfaccia *IUnknown* dell'oggetto un puntatore alla propria interfaccia *IDispatch*. *IDispatch* tiene traccia internamente dei metodi e delle proprietà tramite un identificatore di dispatch (dispID), che è un numero di identificazione univoco per un membro dell'interfaccia. Attraverso *IDispatch* un controller recupera le informazioni di tipo dell'oggetto per l'interfaccia dispatch e quindi collega i nomi dei membri dell'interfaccia ai relativi dispID. Questi dispID sono disponibili in fase di esecuzione e i controller li ottengono chiamando il metodo *GetIDsOfNames* di *IDispatch*.

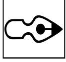
Una volta in possesso del dispID, il controller può chiamare il metodo *Invoke*, di *IDispatch* per eseguire il codice appropriato (proprietà o metodo), compattando i parametri per la proprietà o metodo in uno dei parametri di *Invoke*. *Invoke* ha una segnatura fissata in fase di compilazione che gli consente di accettare un qualsiasi numero di argomenti quando effettua una chiamata a un metodo dell'interfaccia.

L'implementazione di *Invoke* di un oggetto Automation deve quindi decompattare i parametri, chiamare la proprietà o il metodo e stare pronta a gestire gli eventuali errori che si potrebbero verificare. Al ritorno dalla proprietà o dal metodo, l'oggetto passa il suo valore di ritorno al controller.

Questo modo di agire viene detto binding differito perché il controller esegue il binding alla proprietà o al metodo in fase di esecuzione invece che in fase di compilazione.



C++Builder non è in grado di creare CoClass che smistano interfacce non duali. La ragione è che il suo supporto COM si appoggia ad ATL, il quale non supporta interfacce dispatch non duali.



Quando si importa una libreria di tipi, C++Builder richiederà i dispID nel momento in cui genera il codice, permettendo pertanto alle classi dei wrapper generate di chiamare *Invoke* senza chiamare *GetIDsOfNames*. Ciò può incrementare notevolmente le prestazioni di esecuzione dei controller.

Interfacce custom

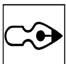
Le interfacce custom sono interfacce definite dall'utente, che consentono ai client di chiamare i metodi dell'interfaccia in base al loro ordine nella VTable e alla conoscenza dei tipi degli argomenti. La VTable elenca gli indirizzi di tutte le proprietà e i metodi che sono membri dell'oggetto, comprese le funzioni membro dell'interfaccia che essa supporta. Se l'oggetto non supporta *IDispatch*, gli elementi relativi ai membri delle interfacce custom dell'oggetto sono elencati nella VTable subito dopo i membri di *IUnknown*.

Se l'oggetto possiede una libreria di tipi, è possibile accedere all'interfaccia custom mediante il layout della sua VTable, che è ottenibile utilizzando il Type Library editor. Se l'oggetto possiede una libreria di tipi e supporta anche *IDispatch*, un client può anche ottenere i dispID dell'interfaccia IDispatch ed eseguire il bind direttamente a un offset di VTable. Il programma per l'importazione delle librerie di tipi di C++Builder (TLIBIMP) recupera i dispID al momento dell'importazione, e pertanto i client che utilizzano i wrapper di dispinterface possono evitare di chiamare GetIDsOfNames; l'informazione è già inclusa nel file _TLB. Tuttavia, i client devono sempre chiamare *Invoke*.

Marshaling dei dati

Per i server out-of-process e i server remoti, occorre tener conto di come COM esegue il marshaling dei dati esternamente al processo corrente. È possibile fornire il marshaling:

- In modo automatico, utilizzando l'interfaccia *IDispatch*.
- In modo automatico, creando una libreria di tipi con il proprio server e contrassegnando l'interfaccia con il flag di OLE Automation. COM sa come eseguire il marshaling di tutti i tipi **compatibili con Automation** presenti nella libreria di tipi e impostare automaticamente i proxy e gli stub. Per consentire il marshaling automatico si applicano alcune restrizioni di tipo.
- In modo manuale implementando tutti i metodi dell'interfaccia *IMarshal*. Questa modalità viene chiamata **marshaling personalizzato**.



Il primo metodo (che usa *IDispatch*) è l'unico disponibile nei server Automation. Il secondo metodo è automaticamente disponibile su tutti gli oggetti che sono stati creati con un wizard e che utilizzano una libreria di tipi.

Tipi compatibili con Automation

Il risultato delle funzioni e i tipi dei parametri dei metodi dichiarati nelle interfacce duali e dispatch e le interfacce contrassegnate come OLE Automation devono essere di tipo *compatibile con Automation*. I tipi elencati di seguito sono di tipo compatibile con OLE Automation:

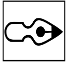
- I tipi validi predefiniti quali *short*, *int*, *single*, *double*, *WideString*. Per un elenco completo, consultare “Tipi consentiti” a pagina 39-12.
- I tipi enumerativi definiti in una libreria di tipi. I tipi enumerativi compatibili con OLE Automation vengono memorizzati come valori a 32-bit e vengono trattati come valori di tipo Integer in modo da consentire il passaggio dei parametri.
- I tipi interfaccia, definiti in una libreria di tipi, che sono innocui per OLE Automation, vale a dire, derivati da *IDispatch* e contenenti solo tipi compatibili con OLE Automation.
- I tipi dispinterface definiti in una libreria di tipi.
- Qualsiasi tipo record custom definito all’interno della libreria di tipi.
- *IFont*, *IStrings*, e *IPicture*. Gli oggetti helper devono essere istanziati in modo da poter mappare
  - un *IFont* to con un *TFont*
  - un *IStrings* con un *TStrings*
  - un *IPicture* con un *TPicture*

  All’occorrenza, i wizard ActiveX control e ActiveForm, creano questi oggetti helper automaticamente. Per utilizzare gli oggetti helper, chiamare le routine globali, rispettivamente *GetOleFont*, *GetOleStrings*, e *GetOlePicture*.

Restrizioni di tipo per il marshaling automatico

Affinché un’interfaccia possa supportare il marshaling automatico (detto anche Automation marshaling o marshaling della libreria di tipi) deve essere soggetta alle seguenti limitazioni. Quando si esegue l’editing dell’oggetto con l’editor della libreria di tipi, l’editor applica le seguenti limitazioni:

- I tipi devono essere compatibili per poter comunicare tra piattaforme differenti. Ad esempio, non è possibile usare strutture dati (se non implementando un altro oggetto proprietà), argomenti senza segno, stringhe AnsiStrings, e così via.
- I tipi di dati String devono essere trasferiti come BSTR. PChar e AnsiString non possono essere smistati con sicurezza.
- Tutti i membri di un’interfaccia duale devono passare un HRESULT come valore di ritorno di una funzione.
- I membri di un’interfaccia duale che restituiscono altri valori devono specificare l’ultimo parametro come **var** o **out**, per indicare un parametro di output che restituisce il valore della funzione.



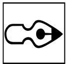
Un modo per superare le restrizioni di tipo dettate da Automation consiste nell'implementare separatamente un'interfaccia *IDispatch* e un'interfaccia custom. In questo modo è possibile usare l'intera gamma dei possibili tipi di argomento. Ciò significa che i client COM hanno la possibilità di usare l'interfaccia custom, a cui i controller Automation possono comunque accedere. In questo caso, tuttavia, il codice per effettuare il marshaling deve essere implementato manualmente.

Marshaling personalizzato

Di solito si usa il marshaling automatico nei server out-of-process e in quelli remoti poiché è molto più semplice-COM provvede a ciò in modo automatico. Tuttavia, si potrebbe voler implementare un marshaling personalizzato nel caso si pensi che ciò possa incrementare le prestazioni del marshaling. Quando si implementa marshaling personalizzato, è necessario supportare l'interfaccia *IMarshal*. Per maggiori informazioni su questo approccio, consultare la documentazione di Microsoft.

Registrazione di un oggetto COM

È possibile registrare l'oggetto server come server in-process o out-of-process. Per informazioni sugli altri tipi di server, consultare il paragrafo"Server in-process, out-of-process e remoti" a pagina 38-6.



Prima di cancellare un oggetto COM dal sistema, occorre rimuoverne la registrazione.

Registrazione di un server in-process

Per registrare un server in-process (DLL or OCX):

- Scegliere il comando Run | Register ActiveX Server.

Per annullare la registrazione di un server in-process:

- Scegliere il comando Run | Unregister ActiveX Server.

Registrazione di un server out-of-process

Per registrare un server out-of-process:

- Eseguire il server con l'opzione della riga di comando /**regserver**.

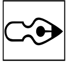
  È possibile impostare le opzioni della riga di comando tramite la finestra di dialogo Run | Parameters.

  È anche possibile registrare il server mandandolo in esecuzione.

Per annullare la registrazione di un server out-of-process:

- Eseguire il server con l'opzione della riga di comando /**unregserver**.

In alternativa, si può usare il comando **tregsvr** dalla riga comandi o eseguire regsvr32.exe dal sistema operativo.



Se il server COM deve essere utilizzato sotto COM+, è necessario installarlo come applicazione COM+ invece di registrarlo. L'installazione dell'oggetto in un'applicazione COM+ provvede a gestire automaticamente la registrazione. Per informazioni su come installare un oggetto in un'applicazione COM+, vedere "Installazione di oggetti transazionali" a pagina 44-29.

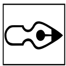
Verifica e debug dell'applicazione

Per eseguire la verifica e il debug dell'applicazione server COM:

1 Se occorre, attivare l'opzione relativa alle informazioni di debug presente sulla pagina Compiler della finestra di dialogo richiamata con Project | Options. Inoltre, attivare l'opzione Integrated Debugging nella finestra di dialogo richiamata con Tools | Debugger Options.
2 Per un server in-process, selezionare il comando Run | Parameters, scrivere nella casella Host Application il nome del controller Automation e scegliere OK.
3 Selezionare il comando Run | Run.
4 Impostare i breakpoint nel server Automation.
5 Utilizzare il controller Automation per interagire con il server Automation.

L'Automation server si fermerà non appena si raggiungeranno i breakpoint.

Inoltre, può essere utile tracciare le chiamate che l'applicazione effettua alle interfacce. Esaminando il flusso delle chiamate COM, è possibile determinare se l'applicazione si comporta secondo le previsioni. Per indicare a C++Builder di aggiungere messaggi nel registro degli eventi ogni volta che vene chiamata un'interfaccia COM, utilizzare la pagina ATL della finestra di dialogo Project objects per specificare le opzioni di debugging.



Come approccio alternativo, nel caso si stia scrivendo anche un Automation controller, è possibile eseguire il debug in un server in-process attivando il supporto cross-process di COM. Utilizzare la pagina General della finestra di dialogo Tools | Debugger Options per attivare il supporto cross-process.

Capitolo

42

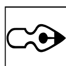
Creazione di Active Server Page

Se si sta utilizzando Microsoft Internet Information Server (IIS) per servire le pagine Web, è possibile utilizzare la tecnologia Active Server Pages (ASP) per creare applicazioni Web client-server dinamiche. La tecnologia Active Server Pages consente di scrivere uno script che viene chiamato ogni volta che il server carica la pagina Web. Questo script, a sua volta, può chiamare oggetti Automation per ottenere informazioni che includerà in una pagina HTML generata. Ad esempio, è possibile scrivere un server Automation in C++Builder, per creare una bitmap o per effettuare una connessione a un database e usare questo controllo per accedere a dati che vengono aggiornati ogni volta che il server carica la pagina Web.

Sul lato client, un documento ASP si comporta come un documento HTML standard e può essere visualizzato dagli utenti su qualsiasi piattaforma utilizzando un qualsiasi Web Browser.

Le applicazioni ASP sono analoghe alle applicazioni che sono scritte utilizzando la tecnologia Web Broker di C++Builder. Per maggiori informazioni sulla tecnologia Web broker, consultare il Capitolo 32, "Creazione di applicazioni server per Internet". Tuttavia, ASP si differenzia per il modo in cui separa il progetto dell'interfaccia utente dall'implementazione delle regole di gestione o dalla complessa logica dell'applicazione.

- Il progetto UI è gestito da Active Server Page. Questo è essenzialmente un documento HTML, ma può inglobare script che chiamano Active Server Objects per fornirgli contenuti che sono il risultato delle regole di gestione o della logica dell'applicazione.
- La logica dell' applicazione è incapsulata da Active Server Objects che espone ad Active Server Page semplici metodi, fornendogli i contenuti necessari.



Sebbene ASP offra il vantaggio di separare il progetto UI dalla logica dell'applicazione, su vasta scala le sue prestazioni risultano limitate. Per quei siti web che rispondono ad un vasto numero di client, si raccomanda invece un approccio basato sulla tecnologia Web Broker.

Lo script delle Active Server Page e gli oggetti Automation che vengono incorporati in un Active Server Page possono fare uso degli intrinsics di ASP (oggetti interni che forniscono informazioni sull'applicazione corrente, sui messaggi HTTP provenienti dal browser e così via)).

Questo capitolo mostra come creare un Active Server Object utilizzando il wizard Active Server Object di C++Builder. Questo speciale controllo Automation può essere chiamato quindi da una Active Server Page e rifornito di contenuti.

Di seguito sono elencati i passi necessari per creare un oggetto Active Server Object:

- Creare un oggetto Active Server Object per l'applicazione.
- Definire l'interfaccia dell'Active Server Object.
- Registrare l'Active Server Object.
- Collaudare ed eseguire il debug dell'applicazione.

Creazione di Active Server Object

Un Active Server Object è un oggetto Automation che ha accesso a informazioni sull'intera applicazione ASP intera e messaggi HTTP utilizzati per comunicare con i browser. Deriva da *TASPObject* o da *TMTSASPObject* (oltre che dalle classi base ATL *CComObjectRootEx* e *CComCoClass*), e supporta protocolli Automation, esponendo sé stesso affinché possa essere utilizzato da altre applicazioni (o dallo script nell'Active Server Page). Si crea un Active Server Object utilizzando il wizard Active Server Object.

Il progetto Active Server Object può essere un eseguibile (exe) o una libreria (dll), a seconda delle proprie necessità. Comunque, è bene essere consapevoli degli inconvenienti derivanti dall'utilizzo di un server out-of-process. Questi inconvenienti sono trattati in "Creazione di ASP per server in-process o out-of-process" a pagina 42-7.

Per visualizzare il wizard Active Server Object:

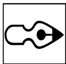
1. Scegliere il comando File | New | Other.
2. Selezionate la pagina identificata con ActiveX.
3. Fare doppio clic sull'icona Active Server Object.

Nel wizard, assegnare un nome al nuovo Active Server Object e specificare il modello di threading che si desidera supportare. Si dovrà scrivere l'implementazione in modo che sia conforme al modello (ad esempio, evitando conflitti di thread). Il modello di threading coinvolge le stesse scelte che si fanno per altri oggetti COM. Per maggiori informazioni, consultare "Scelta di una modello di threadingl" a pagina 41-5.

La cosa che rende unico un Active Server Object è la sua capacità di accedere alle informazioni sull'applicazione ASP e sui messaggi HTTP che passano tra l'Active Server Page e Web browser dei client. Queste informazioni vengono ottenute utilizzando gli intrinsics di ASP. Nel wizard è possibile specificare come l'oggetto accede ad essi impostando il parametro Active Server Type:

- Se si opera con IIS 3 o IIS 4, si utilizzerà Page Level Event Methods. In base a questo modello, l'oggetto implementa i metodi *OnStartPage* e *OnEndPage* che vengono chiamati al momento del caricamento e dello scarico dell'Active Server Page. Quando l'oggetto viene caricato, ottiene automaticamente un'interfaccia *IScriptingContext* che esso utilizza per accedere agli intrinsics di ASP. Queste interfacce, a loro volta, sono rese accessibili come proprietà ereditate dalla classe base (TASPObject).
- Se si opera con IIS5 o con una versione successiva, si utilizza il tipo Object Context. In base a questo modello, l'oggetto recupera un'interfaccia *IObjectContext* che utilizzerà per accedere agli intrinsics di ASP. Ancora una volta, queste interfacce sono rese accessibili come proprietà nella classe base ereditata (*TMTSASPObject*).Uno dei vantaggi derivanti da questo secondo approccio è che l'oggetto ha accesso a tutti gli altri servizi disponibile tramite *IObjectContext*. Per accedere all'interfaccia *IObjectContext*, utilizzare la proprietà *ObjectContext* dell'Active Server Object. Per maggiori informazioni sui servizi disponibili tramite *IObjectContext*, consultare il Capitolo 44, "Creazione di oggetti MTS o COM+".

È possibile chiedere al wizard di generare una semplice pagina ASP che ospiti il nuovo Active Server Object. La pagina generata fornisce uno script ridotto (stilato in VBScript) che crea l'Active Server Object in base al suo ProgID, e indica dove è possibile chiamarne i metodi. Questo script chiama **Server.CreateObject** per avviare l'Active Server Object.



Sebbene lo script di test generato utilizzi VBScript, le Active Server Pages possono anche essere scritte utilizzando Jscript.

All'uscita dal wizard, al progetto corrente viene aggiunta una nuova unit che contiene la definizione dell'Active Server Object. Inoltre, il wizard aggiunge un progetto di libreria di tipi e apre il Type Library editor. A questo punto è possibile esporre le proprietà e i metodi dell'interfaccia mediante la libreria di tipi, come descritto nella sezione "Definizione dell'interfaccia di un oggetto COM" a pagina 41-9. Durante la scrittura dell'implementazione delle proprietà e dei metodi dell'oggetto, è possibile sfruttare gli intrinsics di ASP (descritti di seguito) per ottenere informazioni sull'applicazione ASP e sui messaggi HTTP utilizzati per comunicare con i browser.

Gli Active Server Object, come tutti gli altri oggetti Automation, implementano una **interfaccia duale**, che supporta sia il binding anticipato (in compilazione) tramite la Vtable sia il binding differito (in esecuzione) tramite l'interfaccia *IDispatch*. Per informazioni sulle interfacce duali, consultare "Interfacce duali" a pagina 41-13.

Utilizzo degli intrinsics di ASP

Gli intrinsics di ASP sono una serie di oggetti COM messi a disposizione da ASP agli oggetti che vengono eseguiti in una Active Server Page. Essi consentono all'Active Server Object di accedere alle informazioni relative ai messaggi che intercorrono tra l'applicazione e il Web browser, oltre a fornire un luogo per memorizzare le informazioni condivise tra Active Server Objects appartenenti alla stessa applicazione ASP.

Per semplificare l'accesso a questi oggetti, la classe base dell'Active Server Object li rende disponibili sotto forma di proprietà. Per una conoscenza completa di questi oggetti, consultare la documentazione di Microsoft. I paragrafi seguenti forniscono una breve panoramica.

Application

All'oggetto Application si accede tramite un'interfaccia *IApplicationObject*. Esso rappresenta l'intera applicazione ASP, definita come l'insieme di tutti i file .asp contenuti in una directory virtuale e nelle sue subdirectory. L'oggetto Application può essere condiviso da più client, e pertanto include il supporto per i blocchi che si dovrebbe utilizzare per prevenire conflitti di thread.

IApplicationObject include quanto segue:

Tabella 42.1 Membri dell'interfaccia IApplicationObject

Proprietà, metodo o evento	Significato
Proprietà Contents	Elenca tutti gli oggetti aggiunti all'applicazione utilizzando comandi dello script. Questa interfaccia ha due metodi, *Remove* e *RemoveAll* che è possibile utilizzare per cancellare uno o tutti gli oggetti dall'elenco.
Propietà StaticObjects	Elenca tutti gli oggetti aggiunti all'applicazione usando il marcatore <OBJECT>.
Metodo Lock	Impedisce ad altri client di bloccare l'oggetto Application finché non si chiama Unlock. Tutti i client dovrebbero chiamare Lock prima di accedere alla memoria condivisa (come le proprietà).
Metodo Unlock	Libera tutti i blocchi impostati utilizzando il metodo Lock.
Evento Application_OnEnd	Si verifica quando l'applicazione termina, dopo l'evento Session_OnEnd. Gli unici intrinscs disponibili sono Application e Server. Il gestore di evento deve essere scritto in VBScript o JScript.
Evento Application_OnStart	Si verifica prima che venga creata una nuova sessione (prima di Session_OnStart). Gli unici intrinscs disponibili sono Application e Server. Il gestore di evento deve essere scritto in VBScript o JScript.

Request

All'oggetto Request si accede tramite un'interfaccia *IRequest*. Esso fornisce informazioni sui il messaggio di richiesta HTTP che hanno provocato l'apertura dell'Active Server Page.

IRequest include quanto segue:

Tabella 42.2 Membri dell'interfaccia IRequest

Proprietà, metodo o evento	Significato
Proprietà ClientCertificate	Indica i valori di tutti i campi nel certificato del client trasmesso col messaggio HTTP.
Proprietà Cookies	Indica i valori di tutti gli header Cookie nel messaggio HTTP.
Proprietà Form	Indica i valori degli elementi della scheda nel corpo HTTP. A questi elementi è possibile accedere in base al nome.

Tabella 42.2 Membri dell'interfaccia IRequest (continua)

Proprietà, metodo o evento	Significato
Proprietà QueryString	Indica i valori di tutte le variabili nella stringa della query dall'header HTTP.
Proprietà ServerVariables	Indica i valori di alcune variabili di ambiente. Queste variabili rappresentano le più comuni variabili di header HTTP.
TotalBytes property	Indica il numero di byte nel corpo della richiesta. Questo è il limite superiore sul numero di byte restituito dal metodo BinaryRead.
Metodo BinaryRead	Recupera il contenuto di un messaggio Post. Chiamare il metodo specificando il numero massimo di byte da leggere. Il contenuto risultante viene restituito sotto forma di array Variant di byte. Dopo aver chiamato BinaryRead, non è possibile utilizzare la proprietà Form.

Response

All'oggetto Response si accede tramite un'interfaccia *IResponse*. Consente di specificare le informazioni sul messaggio di risposta HTTP restituito al browser client.

IResponse include quanto segue:

Tabella 42.3 Membri dell'interfaccia IResponse

Proprietà, metodo o evento	Significato
Proprietà Cookies	Determina i valori di tutti gli header Cookie nel messaggio HTTP.
Buffer property	Indica se l'output della pagina è bufferizzato. Se l'output della pagina viene bufferizzato, il server non trasmette una risposta al client finché non sono stati elaborati tutti gli script del server sulla pagina corrente.
Proprietà CacheControl	Determina se i server proxy possono mettere in cache l'output nella risposta.
Proprietà Charset	Aggiunge il nome del set di caratteri all'header del tipo di contenuti.
Proprietà ContentType	Specifica il tipo di contenti HTTP del corpo del messaggio di risposta.
Proprietà Expires	Specifica per quanto tempo la risposta può essere tenuta in cache da un browser prima che scada.
Proprietà ExpiresAbsolute	Specifica la data e l'ora di scadenza della risposta.
IsClientConnected proprietà	Indica se il client si è disconnesso dal server.
Proprietà Pics	Imposta il valore del campo immagine-etichetta dell'header di risposta.
Proprietà Status	Indica lo stato della risposta. Questo è il valore di un header di stato HTTP.
Metodo AddHeader	Aggiunge un header HTTP con nome e valore specificati.
Metodo AppendToLog	Aggiunge una stringa alla fine del file di registro di un server web relativo a questa richiesta.
Metodo BinaryWrite	Scrive informazioni grezze (non interpretate) nel corpo del messaggio di risposta.

Tabella 42.3 Membri dell'interfaccia IResponse (continua)

Proprietà, metodo o evento	Significato
Metodo Clear	Cancella qualsiasi output HTML bufferizzato.
Metodo End	Interrompe l'elaborazione del file .asp e restituisce il risultato corrente.
Metodo Flush	Trasmette immediatamente qualsiasi output bufferizzato.
Metodo Redirect	Trasmette un messaggio di risposta di tipo redirect, reindirizzando il browser client su un URL diverso.
Metodo Write	Scrive una variabile nell'output HTTP corrente sotto forma di stringa.

Session

All'oggetto Session si accede tramite l'interfaccia *ISessionObject*. Consente di memorizzare variabili che permangono per tutta la durata dell'interazione di un client con l'applicazione ASP. Ovvero, queste variabili non vengono azzerate quando il client si sposta da una pagina all'altra all'interno dell'applicazione ASP, ma solamente quando il client esce completamente dall'applicazione.

ISessionObject include quanto segue:

Tabella 42.4 Membri dell'interfaccia ISessionObject

Proprietà, metodo o evento	Significato
Proprietà Contents	Elenca tutti gli oggetti aggiunti alla sessione utilizzando il marcatore <OBJECT>. È possibile accedere a qualsiasi variabile nell'elenco tramite nome, oppure chiamare i metodi *Remove* o *RemoveAll* dell'oggetto Contents per cancellare dei valori.
Propietà StaticObjects	Elenca tutti gli oggetti aggiunti alla sessione con il marcatore <OBJECT>.
Proprietà CodePage	Specifica la code page da utilizzare per la mappatura dei simboli. Impostazioni locale diverse possono utilizzare code page diverse.
Proprietà LCID	Specifica l'identificatore dell'impostazione locale da utilizzare per interpretare il contenuto della stringa.
Proprietà SessionID	Indica l'identificatore di sessione per il client attuale.
Proprietà TimeOut	Specifica per quanto tempo, in minuti, permane la sessione senza che vi sia una richiesta (o un aggiornamento) da parte client fino al termine dell'applicazione.
Metodo Abandon	Distrugge la sessione e ne libera le risorse.
Evento Session_OnEnd	Si verifica quando la sessione vie abbandonata o va in time-out. Gli unici intrinsics disponibili sono Application, Server e Session. Il gestore di evento deve essere scritto in VBScript o JScript.
Evento Session_OnStart	Si verifica quando il server crea una nuova sessione (dopo Application_OnStart ma prima di eseguire lo script sull'Active Server Page). Tutti gli intrinsics sono disponibili. Il gestore di evento deve essere scritto in VBScript o JScript.

Server

All'oggetto Server si accede tramite un'interfaccia *IServer*. Fornisce vari strumenti di utilità per scrivere l'applicazione ASP.

IServer include quanto segue:

Tabella 42.5 Membri dell'interfaccia IServer

Proprietà, metodo o evento	Significato
Proprietà ScriptTimeOut	Analogo alla proprietà TimeOut dell'oggetto Session.
Metodo CreateObject	Istanzia uno specifico Active Server Object.
Metodo Execute	Esegue lo script in uno specifico file .asp.
Metodo GetLastError	Restituisce un oggetto ASPError che descrive la condizione di errore.
Metodo HTMLEncode	Codifica una stringa per utilizzarla in un header HTML, sostituendo caratteri riservati opportune costanti simboliche.
Metodo MapPath	Mappa un determinato percorso virtuale (un percorso assoluto sul server corrente o un percorso relativo alla pagina corrente) in un percorso fisico.
Metodo Transfer	Trasmette tutte le informazioni di stato correnti a un'altra Active Server Page affinché le elabori.
Metodo URLEncode	Applica le regole di codifica degli URL, compresi i caratteri di escape, a una stringa specificata.

Creazione di ASP per server in-process o out-of-process

È possibile utilizzare **Server.CreateObject** in una pagina ASP per avviare server in-process o out-of process, in base alle proprie necessità. Comunque, l'avvio di un server in process è la scelta più comune.

A differenza della maggior parte dei server in-process, un Active Server Object in un server in-process non viene eseguito nello spazio di processo del client. Viene invece eseguito nello spazio di processo di IIS. Ciò significa che il client non deve scaricare l'applicazione (come fa, per esempio, quando si utilizzano oggetti ActiveX). Le DLL di componenti in-process sono più rapide e sicure dei server out-of-process, risultando così più adatte per l'uso sul lato server.

Poiché i server out-of-process sono meno sicuri, di solito IIS viene configurato per *non* ammettere eseguibili out-of-process. In questo caso, la creazione di un server out-of-process per l'Active Server Object potrebbe generare un errore simile al seguente:

```
Server object error 'ASP 0196'
Cannot launch out of process component
/path/outofprocess_exe.asp, line 11
```

Inoltre, i componenti out-of-process spesso creano singoli processi server per ciascuna istanza dell'oggetto, risultando così più lenti delle applicazioni CGI. Non sono scalabili altrettanto bene come i componenti DLL.

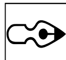
Se le prestazione e la scalabilità sono requisiti prioritari del proprio sito, si raccomanda l'uso di server in-process.Comunque, i siti Intranet che hanno un traffico

da moderato a scarso potrebbero utilizzare un componente out-of-process senza influenzare in modo negativo le prestazioni globali del sito.

Per informazioni generali sui server in-process e out-of-process, consultare, “Server in-process, out-of-process e remoti”, a pagina 38-6.

Registrazione di Active Server Object

È possibile registrare un oggetto Active Server Page sia come server in-process sia come server out-of-process. Comunque, i server in-process sono i più comuni.



Se si desidera rimuovere dal sistema l’oggetto Active Server Page, si dovrebbe per prima cosa annullare la registrazione, eliminando dal file di registro di Windows gli elementi ad esso relativi.

Registrazione di un server in-process

Per registrare un server in-process (DLL or OCX):

- Scegliere il comando Run | Register ActiveX Server.

Per annullare la registrazione di un server in-process:

- Scegliere il comando Run | Unregister ActiveX Server.

Registrazione di un server out-of-process

Per registrare un server out-of-process:

- Eseguire il server usando l’opzione /regserver della riga comandi. (È possibile impostare le opzioni della riga comandi mediante la finestra di dialogo richiamando tramite il comando Run | Parameters)

  È anche possibile registrare il server mandandolo in esecuzione.

Per annullare la registrazione di un server out-of-process:

- Eseguire il server usando l’opzione /unregserver della riga comandi.

Collaudo e debug dell’applicazione Active Server Page

Il debug di un qualsiasi server in-process, come un Active Server Object, è simile al debug di una DLL. Si deve scegliere un’applicazione host che carichi la DLL e quindi eseguire il debug come al solito. Per collaudare ed eseguire il debug di un oggetto Active Server Object:

1 Se occorre, attivare l’opzione relativa alle informazioni di debug presente sulla pagina Compiler della finestra di dialogo richiamata con Project | Options. Inoltre,

attivare l'opzione Integrated Debugging nella finestra di dialogo richiamata con Tools|Debugger Options.

2 Selezionare il comando Run|Parameters, scrivere il nome del server web nella casella Host Application e scegliere OK.

3 Selezionare il comando Run|Run.

4 Impostare i breakpoint nell'implementazione dell'Active Server Object.

5 Utilizzare il browser Web per interagire con l'Active Server Page.

Il debugger si interrompe non appena viene raggiunto un breakpoint.

Capitolo

43

Creazione di un controllo ActiveX

Un controllo ActiveX è un componente software che integra ed estende le funzionalità di una qualsiasi applicazione host in grado di supportare i controlli ActiveX, come C++Builder, Delphi, Visual Basic, Internet Explorer e (dato un plug-in) Netscape Navigator. I controlli ActiveX implementano un particolare set di interfacce che consentono questa integrazione.

C++Builder, per esempio, viene fornito con diversi controlli ActiveX, che comprendono diagrammi, fogli elettronici e controlli grafici. È possibile aggiungere questi controlli alla tavolozza componenti nell'IDE, e quindi usarli come qualsiasi componente VCL standard, trascinandoli sulle schede e impostandone le proprietà tramite l'Object Inspector.

Un controllo ActiveX può essere distribuito anche su Web, consentendo che possa essere referenziato nei documenti HTML e visualizzato con i browser Web abilitati per ActiveX.

C++Builder fornisce wizard che consentono di creare due tipi di controlli:

- **Controlli ActiveX che avvolgono classi VCL**. Avvolgendo una classe VCL, è possibile convertire componenti esistenti in controlli ActiveX o crearne di nuovi, collaudarli localmente, e quindi convertirli in controlli ActiveX. I controlli ActiveX sono pensati di solito per essere incorporati in una applicazione host più grande.
- **Active form**. Le Active form consentono di utilizzare il modulo di progettazione delle schede per creare un controllo più elaborato che agisce come una finestra di dialogo o come un'applicazione completa. Una Active form viene sviluppata in modo molto simile a una tipi applicazione C++Builder. Le Active form sono pensate di solito per la distribuzione su Web.

Questo capitolo fornisce una panoramica su come creare un controllo ActiveX nell'ambiente C++Builder. Lo scopo del capitolo non è quello di fornire tutti i dettagli implementativi per la scrittura di un controllo ActiveX senza fare ricorso a un wizard. Per tali informazioni, fare riferimento alla documentazione Microsoft Developer's Network (MSDN) o ricercare sul sito web di Microsoft le informazioni relative agli ActiveX.

Panoramica sulla creazione di controlli ActiveX

La creazione di controlli ActiveX con C++Builder è molto simile alla creazione dei controlli o delle schede. L'operazione differisce sostanzialmente dalla creazione di altri oggetti COM, in cui per prima cosa si definisce l'interfaccia dell'oggetto e quindi si completa l'implementazione. Per creare controlli ActiveX (diversi dalle Active form), il processo viene invertito, cominciando con l'implementazione di un controllo VCL e quindi generando l'interfaccia e la libreria di tipi una volta scritto il controllo. Quando si creano Active forms, l'interfaccia e la libreria di tipi vengono create contemporaneamente alla scheda, e in una fase successiva si utilizza il modulo di progettazione della scheda per implementare la scheda.

Il controllo ActiveX completato è composto da un controllo VCL, che fornisce l'implementazione sottostante, un oggetto COM che avvolge il controllo VCL e una libreria di tipi che elenca proprietà, metodi ed eventi dell'oggetto COM.

Per creare un nuovo controllo ActiveX (diverso da un Active Form), seguire i passi seguenti:

1. Progettare e creare il controllo VCL custom che forma la base del controllo ActiveX.
2. Utilizzare il wizard ActiveX control per generare un controllo ActiveX da un controllo VCL creato nel punto 1.
3. Utilizzare il wizard ActiveX property page per creare una o più pagine di proprietà per il controllo (facoltativo).
4. Associare la pagina della proprietà con il controllo ActiveX (facoltativo).
5. Registrare il controllo.
6. Collaudare il controllo con tutte le potenziali applicazioni di destinazione.
7. Distribuire il controllo ActiveX sul Web. (facoltativo)

Per creare un nuovo Active form, compiere le operazioni seguenti:

1. Utilizzare il wizard ActiveForm per creare una Active form, che appare nell'IDE come una scheda vuota, e un wrapper ActiveX associato a quella scheda.
2. Utilizzare Form Designer per aggiungere componenti alla Active form e implementarne il comportamento, esattamente nello stesso modo con cui si crea e si implementatate una comune scheda utilizzando Form Designer.
3. Seguire i passi da 3 a 7 riportati in precedenza per assegnare alla Active form una pagina di proprietà, registrarla e distribuirla sul Web.

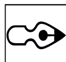
Elementi di un controllo ActiveX

Un controllo ActiveX implica molti elementi, ognuno dei quali compie una funzione specifica. Gli elementi comprendono un controllo VCL, un oggetto wrapper COM corrispondente che esponga proprietà, metodi, eventi e una o più librerie di tipi associate.

Il controllo VCL

L'implementazione alla base di un controllo ActiveX in C++Builder è un controllo della VCL. Quando si crea un controllo ActiveX, per prima cosa è necessario progettare o scegliere il controllo VCL da cui sarà generato il controllo ActiveX.

Il controllo VCL sottostante deve essere un discendente di *TWinControl*, in quanto deve avere una finestra che possa apparentarsi con l'applicazione host. Quando si create una Active form, questo oggetto è un discendente di *TActiveForm*.



Il wizard ActiveX control elenca i discendenti *TWinControl* disponibili fra i quali se ne può sceglierne uno da trasformare in un controllo ActiveX. Questo elenco, comunque non include tutti i discendenti di *TWinControl*. Alcuni controlli, come ad esempio *THeaderControl*, sono registrati come incompatibile con ActiveX (utilizzando la procedura *RegisterNonActiveX*) e non appaiono nell'elenco.

Wrapper ActiveX

L'oggetto COM attuale è un oggetto wrapper ActiveX per il controllo VCL. Ha il nome della scheda TVCLClassXImpl, in cui *TVCLClass* è il nome della classe del controllo VCL. Pertanto, per fare un esempio, il nome del wrapper ActiveX per *TButton* sarebbe *TButtonXImpl*.

La classe wrapper discende dalle classi dichiarate dalla macro VCLCONTROL_IMPL che fornisce il supporto per le interfacce ActiveX. Il wrapper ActiveX eredita questo supporto che gli consente di trasmettere messaggi di Windows al controllo VCL e apparentare la sua finestra nell'applicazione host.

Il wrapper ActiveX espone ai client le proprietà e i metodi del controllo VCL tramite la sua interfaccia predefinita. Il wizard implementa automaticamente la maggior parte delle proprietà e dei metodi della classe wrapper, delegando le chiamate ai metodo al controllo VCL associato. Il wizard fornisce anche alla classe wrapper i metodi che attivano gli eventi del controllo VCL sui client e assegna questi metodi come gestori di evento nel controllo VCL.

La libreria di tipi

I wizard ActiveX control generano automaticamente una libreria di tipi che contiene le definizioni di tipo per la classe wrapper, la sua interfaccia predefinita e tutte le definizioni di tipo richieste. Queste informazioni di tipo forniscono un modo per il controllo per rendere noti i propri servizi alle applicazioni host. È possibile visualizzare e modificare queste informazioni utilizzando il Type Library editor. Sebbene queste informazioni siano memorizzate in un file binario di libreria di tipi separato (estensione .TLB), esse vengono anche compilate automaticamente nel controllo ActiveX DLL sotto forma di risorsa.

Pagina di proprietà

Se si vuole, è possibile dotare il controllo ActiveX di una pagina di proprietà. La pagina di proprietà consente all'utente di un'applicazione host (il client) di vedere e modificare le proprietà del controllo. È possibile raggruppare più proprietà su una sola pagina, oppure utilizza una pagina per fornire un'interfaccia per una determinata proprietà simile a una finestra di dialogo. Per informazioni su come

creare pagine di proprietà, consultare "Creazione di una pagina di proprietà per un controllo ActiveX" a pagina 43-14.

Progettazione di un controllo ActiveX

Quando si progetta un controllo ActiveX, si inizia creando un controllo VCL custom. In questo modo si forma la base del controllo ActiveX. Per informazioni sulla creazione di controlli custom, consultare la Parte V, "Creazione di componenti custom".

Quando si progetta un controllo VCL, è bene ricordare che sarà incorporato in un'altra applicazione; questo controllo non è di per sé un'applicazione. Per questo motivo, molto probabilmente non si utilizzeranno finestre di dialogo elaborate o altri importanti componenti dell'interfaccia utente. Solitamente, il fine ultimo è quello di creare un semplice controllo con un determinato comportamento interno e che segua le regole dell'applicazione principale.

Inoltre, si dovrebbe verificare che i tipi di tutte le proprietà e i metodi che si vuole siano esposti ai client dall'oggetto siano compatibili con Automation, poiché l'interfaccia del controllo ActiveX deve supportare *IDispatch*. Il wizard non aggiunge alcun metodo all'interfaccia di una classe wrapper i cui parametri non siano compatibili con Automation. Per un elenco di tipi compatibili con Automation, consultare "Tipi consentiti" a pagina 39-12.

I wizard implementano tutte le interfacce ActiveX necessarie richieste utilizzando la classe wrapper COM. I wizard implementano tutte le interfacce ActiveX necessarie richieste utilizzando la classe wrapper COM. Inoltre rendono anche accessibili tutte le proprietà, i metodi e gli eventi compatibili con Automation mediante l'interfaccia predefinita della classe wrapper. Una volta che il wizard ha generato la classe wrapper COM e la relativa interfaccia, è possibile utilizzare il Type Library editor per modificare l'interfaccia predefinita o potenziare la classe wrapper implementando interfacce aggiuntive.

Generazione di un controllo ActiveX da un controllo VCL

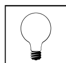
Per generare un controllo ActiveX da un controllo VCL, utilizzare il wizard ActiveX Control. Le proprietà, i metodi e gli eventi del controllo VCL diventano la proprietà, i metodi e gli eventi del controllo ActiveX.

Prima di utilizzare il wizard ActiveX control, si deve decidere quale sarà controllo VCL che dovrà fornire al controllo ActiveX generato l'implementazione di base.

Per attivare il wizard ActiveX control:

1. Scegliere File | New | Other per aprire la finestra di dialogo New Items.
2. Selezionare la pagina ActiveX.
3. Fare doppio clic sull'icona ActiveX Control.

Nel wizard, selezionare il nome del controllo VCL che sarà sviluppato dal nuovo controllo ActiveX. La finestra di dialogo elenca tutti i controlli disponibili, discendenti da *TWinControl* e che non sono stati registrati come incompatibile con ActiveX utilizzando la procedura *RegisterNonActiveX*.



Se nell'elenco a discesa non si vede il controllo desiderato, controllare se è stato installato nell'IDE o se la sua unit è stata aggiunta al progetto.

Una volta selezionato un controllo VCL, il wizard genera automaticamente un nome per la CoClass, la unit di implementazione per il wrapper ActiveX, e il progetto della libreria ActiveX. (Se al momento c'è un progetto di libreria ActiveX aperto, e non contiene un oggetto evento COM+, viene utilizzato automaticamente il progetto corrente). Tutti questi elementi sono modificabili nel wizard (a meno che non vi sia già un progetto di libreria ActiveX aperto, nel qual caso il nome del progetto non è modificabile).

Il wizard specifica sempre Apartment come modello di threading. Questo non è un problema se il progetto ActiveX contiene, come accade di solito, solamente un singolo controllo. Tuttavia, se al progetto si aggiungono altri oggetti, sarà compito del programmatore fornire il supporto per il thread.

Il wizard consente anche di configurare varie opzioni sul controllo ActiveX:

- **Attivazione della licenza**: È possibile rendere il controllo soggetto a licenza in modo da garantirsi che gli utenti del controllo non lo possano aprire per utilizzarlo in altri progetti o in esecuzione a meno che non abbiano una chiave di licenza.
- **Inclusione di informazioni sulla versione**: È possibile includere nel controllo ActiveX informazioni sulla versione, come un copyright o una descrizione del file. Queste informazioni possono essere viste in un browser. Alcuni client host, come Visual Basic 4.0, richiedono le informazioni di versione pena il mancato utilizzo del controllo ActiveX. Specificare informazioni di versione scegliendo il comando Project|Options e selezionando la pagina Version Info.
- **Inclusione di una finestra About**: È possibile chiedere al wizard di generare una scheda separata che implementi una finestra About (Informazioni su) per il controllo. Gli utenti dell'applicazione host possono visualizzare questa finestra About nell'ambiente di sviluppo. Per impostazione predefinita, la finestra About include il nome del controllo ActiveX, un'immagine, informazioni sul copyright e un pulsante OK. in un secondo tempo sarà possibile modificare questa scheda predefinita che il wizard aggiunge al progetto.

All'uscita dal wizard, esso genera quanto segue:

- Un file progetto ActiveX Library, che contiene il codice necessario per avviare un controllo ActiveX. Normalmente non c'è alcuna necessità di modificare questo file.
- Una libreria di tipi che definisce la CoClass del controllo, l'interfaccia che essa espone ai client, e tutte le definizione di tipo richieste. Per maggiori informazioni sulla libreria di tipi, fare riferimento al Capitolo 39, "Funzionamento delle librerie di tipi".
- Una unit di implementazione di ActiveX, che definisce e implementa il controllo ActiveX, adattato alla sintassi ATL (Active Template Library) di Microsoft

utilizzando la macro VCLCONTROL_IMPL. Questo controllo ActiveX è un'implementazione perfettamente funzionante che non richiede ulteriore lavoro. Comunque, se voi desidera personalizzare proprietà, metodi e eventi che il controllo ActiveX espone ai client è possibile modificare questa classe.

- Una unit ATL il cui nome assume il formato *ActiveXControlProj*_ATL.cpp (.h), in cui *ActiveXControlProj* è il nome del progetto. Questa unit è composta principalmente da istruzioni include che rendono disponibili al progetto le classi modello ATL e da definizioni di classi che consentono al wrapper ActiveX generato di operare con un oggetto VCL. Dichiara anche la variabile globale di nome _Module che rappresenta la libreria ActiveX alle classi di ATL.
- Una finestra di dialogo About e una unit nel caso sia stata scelta questa opzione.
- Un file .LIC nel caso si sia attivato il controllo sulla licenza.

Generazione di un controllo ActiveX basato su una scheda VCL

A differenza di altri controlli ActiveX, le Active forms non vengono prima progettate e poi avvolte in una classe wrapper ActiveX. Il wizard ActiveForm genera invece una scheda vuota che sarà progettata alla fine, quando il wizard attiverà Form Designer.

Quando una ActiveForm viene distribuita su Web, C++Builder crea una pagina HTML che contiene un riferimento all'ActiveForm e specifica la sua ubicazione sulla pagina. La ActiveForm può visualizzarla ed eseguirla da un Web browser. Nel browser la scheda si comporta esattamente come una scheda indipendente di C++Builder. La scheda può contenere qualsiasi componente VCL o controllo ActiveX, inclusi eventuali controlli VCL custom.

Per avviare il wizard ActiveForm:

1. Scegliere File|New|Other per aprire la finestra di dialogo New Items.
2. Selezionare la pagina ActiveX.
3. Fare doppio clic sull'icona ActiveForm.

Il wizard Active form assomiglia al wizard ActiveX control, tranne che per il fatto che non è possibile specificare il nome della classe VCL da avvolgere. Questo perché le Active forms sempre sono basate su *TActiveForm*.

Come nel wizard per i controlli ActiveX, è possibile modificare i nomi predefiniti della CoClass, della unit di implementazione e del progetto di libreria ActiveX. Analogamente, il wizard consente di indicare se si desidera che la Active form sia soggetta a licenza, se si devono includere le informazioni di versione e se si desidera una finestra About.

All'uscita dal wizard, viene generato quanto segue:

- Un file di progetto di Libreria ActiveX contenente il codice necessario per avviare un controllo ActiveX. Di solito questo file non viene modificato.
- Una libreria di tipi, che definisce la CoClass del controllo, l'interfaccia che espone ai client, e tutte le definizioni di tipo necessarie. Per maggiori informazioni sulla

libreria di tipi, fare riferimento al Capitolo 39, “Funzionamento delle librerie di tipi”.

- Una scheda che discende da *TActiveForm*. Questa scheda appare nel modulo di impostazione della scheda, in cui sarà possibile progettare in modo visuale la Active form che apparirà ai client. La sua implementazione apparirà nella unit di implementazione generata.
- La dichiarazione di un wrapper ActiveX per la scheda. Questa classe wrapper è definita anche nella unit di implementazione, il cui nome assume il formato TActiveFormXImpl, in cui *TActiveFormX* è il nome della classe scheda. Questo wrapper ActiveX è un’implementazione completamente funzionante che non richiede alcun lavoro aggiuntivo. Comunque, se si desiderano personalizzare le proprietà, i metodi e gli eventi che la Active form espone ai client, è possibile modificare questa classe.
- Una unit ATL il cui nome assume il formato *ActiveXControlProj*_ATL.cpp (.h), in cui *ActiveXControlProj* è il nome del progetto. Questa unit è composta principalmente da istruzioni include che rendono disponibili al progetto le classi modello ATL, e da definizioni di classi che consentono al wrapper ActiveX generato di operare con una scheda VCL. Dichiara anche la variabile globale di nome _Module che rappresenta la libreria ActiveX alle classi di ATL.
- Un scheda About nel caso sia stata richiesta.
- Un file .LIC nel caso si sia attivato il controllo sulla licenza.

A questo punto è possibile aggiungere controlli e progettare la scheda a proprio piacimento.

Dopo aver progettato e compilato il progetto ActiveForm in una libreria ActiveX (che ha estensione OCX), è possibile distribuire il progetto sul server Web e C++Builder crea una pagina HTML di verifica con un riferimento alla ActiveForm appena creata.

Concessione in licenza di controlli ActiveX

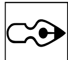
La concessione in licenza di un controllo ActiveX consiste nel preparare una chiave di licenza in fase di progettazione e nel supportare la creazione dinamica delle licenze se il controllo viene creato in fase di esecuzione.

Per preparare le licenze in fase di progettazione, il wizard ActiveX crea una chiave per il controllo che viene registrata in un file avente lo stesso nome del progetto e l’estensione .LIC. Questo file .LIC viene aggiunto al progetto. L’utente del controllo deve possedere una copia del file LIC per aprire il controllo nell’ambiente di sviluppo. Ogni controllo nel progetto, per cui è stata attivata l’opzione Make Control Licensed, avrà nel file LIC una voce di chiave separata.

Per supportare le licenze di esecuzione, la classe wrapper implementa due metodi, *GetLicenseString* e *GetLicenseFilename*. Questi metodi restituiscono rispettivamente la stringa di licenza del controllo e il nome del file .LIC. Quando un’applicazione host tenta di creare il controllo ActiveX, la fabbrica di classi del controllo chiama questi

metodi e paragona la stringa restituita da *GetLicenseString* con la stringa memorizzata nel file .LIC.

Le licenze di esecuzione per Internet Explorer richiedono un ulteriore livello di indirezione perché gli utenti possono visualizzare il codice sorgente HTML di qualunque pagina Web, e perché il controllo ActiveX, prima di essere visualizzato, viene copiato sul computer dell'utente. Per creare licenze di esecuzione per i controlli utilizzati con Internet Explorer, è necessario per prima cosa generare un file package della licenza (file LPK) e incorporare questo file nella pagina HTML che contiene il controllo. Il file LPK è fondamentalmente un array di CLSID di controlli ActiveX e di chiavi di licenza.



Per generare il file LPK, utilizzare il programma di utilità LPK_TOOL.EXE, che è possibile scaricare dal sito Web di Microsoft (www.microsoft.com).

Per incorporare il file LPK in una pagina Web, utilizzare gli oggetti HTML <OBJECT> e <PARAM> nel seguente modo:

```
<OBJECT CLASSID="clsid:6980CB99-f75D-84cf-B254-55CA55A69452">
    <PARAM NAME="LPKPath" VALUE="ctrllic.lpk">
</OBJECT>
```

Il CLSID identifica l'oggetto come un package di licenze e PARAM specifica l'ubicazione relativa del file di package di licenze rispetto alla pagina HTML.

Quando Internet Explorer tenta di visualizzare una pagina Web contenente il controllo, analizza il file .LPK, estrae la chiave di licenza e, se la licenza coincide con la licenza del controllo (restituita da *GetLicenseString*), traccia il controllo sulla pagina. Se nella pagina Web è incluso più di file .LPK, Internet Explorer ignora tutti i file tranne il primo.

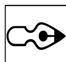
Per maggiori informazioni, cercare "Licensing ActiveX Controls" sul sito Web di Microsoft.

Personalizzazione dell'interfaccia del controllo ActiveX

I wizard ActiveX Control e ActiveForm generano un'interfaccia predefinita per la classe wrapper di ActiveX. Questa interfaccia predefinita espone semplicemente le proprietà, i metodi e gli eventi del controllo o della scheda VCL originale, con le seguenti eccezioni:

- Le proprietà associate ai dati non appaiono. Poiché i controlli ActiveX hanno un meccanismo diverso per rendere i controlli associati ai dati rispetto ai controlli VCL, i wizard non convertono le proprietà relative ai dati. Per informazioni su come rendere data-aware il controllo ActiveX consultare "Attivazione dell'associazione di dati semplici con la libreria di tipi" a pagina 43-12.
- Qualsiasi proprietà, metodo o evento il cui tipo non sia compatibile con Automation non appare. Se lo si desidera, potrebbero essere aggiunti all'interfaccia del controllo ActiveX una volta che il wizard ha terminato le operazioni.

È possibile aggiungere, modificare e rimuovere le proprietà, i metodi e gli eventi di un controllo ActiveX effettuando l'editing della libreria di tipi usando il Type Library editor come descritto nel Capitolo 39, "Funzionamento delle librerie di tipi".

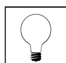


È possibile aggiungere proprietà non pubbliche all'interfaccia del controllo ActiveX. Tali proprietà possono essere impostate in esecuzione e appariranno nell'ambiente di sviluppo, ma le modifiche ad esse apportate non saranno permanenti. Ovvero, se l'utente del controllo modifica il valore di una proprietà in fase di progettazione, le modifiche non saranno visibili nel controllo al momento dell'esecuzione. Questo perché i controlli ActiveX utilizzano il sistema di streaming della VCL invece del sistema di streaming di ATL. Se il sorgente è un oggetto VCL e la proprietà non è stata resa pubblica, è possibile rendere permanenti alcune proprietà creando un discendente dell'oggetto VCL e rendendo pubblica la proprietà nel discendente.

Si potrebbe anche scegliere di non esporre tutte le proprietà, i metodi e gli eventi del controllo VCL alle applicazioni host. È possibile utilizzare il Type Library editor per rimuoverli dalle interfacce generate dal wizard. Quando si rimuovono proprietà e metodi da un'interfaccia utilizzando il Type Library editor, il Type Library editor non li rimuove dalla classe dell'implementazione corrispondente. Modificare la classe del wrapper ActiveX nella unit di implementazione per rimuoverli dopo aver modificato l'interfaccia nel Type Library editor.



Qualsiasi modifica apportata alla libreria di tipi andrà perduta se si rigenera il controllo ActiveX dal controllo VCL o dalla scheda originali.



È una buona idea controllare i metodi che il wizard aggiunge alla classe del wrapper ActiveX. Non solo si ha l'opportunià di controllare se il wizard ha omesso qualche proprietà associata ai dati o metodi non compatibili con Automation, ma si ha la possibilità di lascia di scoprire i metodi per i quali il wizard non è stato in grado di generare un'implementazione. Tali metodi appaiono nell'implementazione con un commento che indica il problema.

Aggiunta di ulteriori proprietà, metodi ed eventi

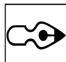
L'aggiunta di proprietà e metodi all'interfaccia del controllo ActiveX è analoga all'aggiunta di proprietà, metodi e eventi ad una qualsiasi interfaccia COM. L'aggiunta di eventi è analoga all'aggiunta di metodi (i gestori di evento), tranne nel caso che li si aggiunga all'interfaccia Events invece che all'interfaccia predefinita dell'oggetto. L'utilizzo del Type Library editor per aggiungere proprietà e eventi è descritto in "Definizione dell'interfaccia di un oggetto COM" a pagina 41-9.

Quando si esegue un'aggiunta all'interfaccia di un controllo ActiveX, spesso si sta solo rendendo visibile una proprietà, un metodo o un evento nel controllo VCL associato. I paragrafi seguenti descrivono come compiere tale operazione.

Aggiunta di proprietà e metodi

La classe del wrapper ActiveX implementa le proprietà nella propria interfaccia utilizzando metodi di accesso per la lettura e la scrittura. Ovvero, la classe del wrapper ha proprietà COM che appaiono in un'interfaccia come metodi getter e/o setter. A differenza delle proprietà VCL, nell'interfaccia delle proprietà COM non si

vedono dichiarazioni di "proprietà". Piuttosto, si vedono metodi che sono indicati come metodi di accesso alle proprietà. Quando si aggiunge una proprietà all'interfaccia predefinita del controllo ActiveX, nella definizione della classe del wrapper (che appare nella unit _TLB, aggiornata dal Type Library editor) vengono aggiunti uno o due nuovi metodi (un setter e/o un getter) che occorre implementare, esattamente come quando, aggiungendo un metodo all'interfaccia, alla classe del wrapper viene aggiunto automaticamente un metodo corrispondente da implementare. Pertanto, l'aggiunta di proprietà all'interfaccia della classe del wrapper è praticamente analoga all'aggiunta di metodi: alla definizione della classe del wrapper vengono aggiunte automaticamente nuove implementazioni scheletriche di metodo che occorre completare.



Per maggiori informazioni su ciò che appare nella unit _TLB generata, consultare "Codice generato durante l'importazione delle informazioni di una libreria di tipi" a pagina 40-5.

Per esempio, si consideri una proprietà *Caption*, di tipo AnsiString nell'oggetto VCL associato. Quando si aggiungete questa proprietà nel Type Library editor, C++Builder aggiunge alla classe del wrapper le seguenti dichiarazioni:

```
STDMETHOD(get_Caption(BSTR* Value));
STDMETHOD(set_Caption(BSTR Value));
```

Inoltre, aggiunge automaticamente implementazioni scheletriche di metodo da completare:

```
STDMETHODIMP TButtonXImpl::get_Caption(BSTR* Value)
{
  try
  {
  }
  catch(Exception &e)
  {
    return Error(e.Message.c_str(), IID_IButtonX);
  }
  return S_OK;
};

STDMETHODIMP TButtonXImpl::set_Caption(BSTR Value)
{
  try
  {
  }
  catch(Exception &e)
  {
    return Error(e.Message.c_str(), IID_IButtonX);
  }
  return S_OK;
};
```

Di solito, è possibile implementare questi metodi semplicemente delegandoli al controllo VCL associato, a cui si può accedere utilizzando il membro *m_VclCtl* della classe del wrapper:

```
STDMETHODIMP TButtonXImpl::get_Caption(BSTR* Value)
{
```

```
  try
  {
    *Value = WideString(m_VclCtl->Caption).Copy();
  }
  catch(Exception &e)
  {
    return Error(e.Message.c_str(), IID_IButtonX);
  }
  return S_OK;
};

STDMETHODIMP TButtonXImpl::set_Caption(BSTR Value)
{
  try
  {
    m_VclCtl->Caption = AnsiString(Value);
  }
  catch(Exception &e)
  {
    return Error(e.Message.c_str(), IID_IButtonX);
  }
  return S_OK;
};
```

In alcuni casi, potrebbe essere necessario aggiungere del codice per convertire i tipi di dati COM in tipi C++ nativi. L'esempio precedente gestisce questa conversione di tipo.

Aggiunta di eventi

Il controllo ActiveX può scatenare eventi nel suo contenitore nello stesso modo in cui un oggetto Automation scatena eventi nei client. Questo meccanismo è descritto in "Esposizione di eventi ai client" a pagina 41-11.

Se il controllo VCL utilizzato come base del controllo ActiveX non ha alcun evento pubblicato, i wizard aggiungono automaticamente il supporto necessario per gestire un elenco di eventi del client da convogliare alla classe del wrapper ActiveX e definiscono la dispinterface uscente che i client devono implementare per rispondere agli eventi.

Per scatenare eventi sul contenitore, la classe del wrapper ActiveX deve implementare un gestore di evento per l'evento sull'oggetto VCL. Questo gestore di evento chiama il metodo *Fire_EventName*, implementato dalla classe TEvents_*CoClassName* definita nella unit _TLB:

```
void __fastcall TButtonXImpl::KeyPressEvent(TObject *Sender, char &Key)
{
  short TempKey;
  TempKey = (short)Key;
  Fire_OnKeyPress(&TempKey);
  Key = (short)TempKey;
};
```

Si dovrà in seguito assegnare questo gestore di evento al controllo VCL in modo che venga chiamato al verificarsi dell'evento. Per fare ciò, aggiungere l'evento al metodo

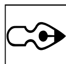
InitializeControl che appare nella dichiarazione della classe del wrapper (nella header della unit di implementazione):

```
void InitializeControl()
{
  m_VclCtl->OnClick = ClickEvent;
  m_VclCtl->OnKeyPress = KeyPressEvent;
}
```

Attivazione dell'associazione di dati semplici con la libreria di tipi

Con un semplice bind di dati è possibile associare una proprietà del controllo ActiveX ad un determinato campo di un database. Per fare ciò, il controllo ActiveX deve comunicare all'applicazione host quale valore rappresenta i dati del campo e quando vengono modificati. Questa comunicazione si stabilisce impostando i flag di binding della proprietà con il Type Library editor.

Se si contrassegna una proprietà come bindable, quando un utente modifica la proprietà (quale un campo di un database), il controllo notifica al contenitore (l'applicazione host client) che il valore è stato modificato e richiede che il record del database venga aggiornato. Il contenitore interagisce col database e notifica quindi al controllo se l'aggiornamento del record è andato a buon fine o meno.



L'applicazione contenitore che ospita il controllo ActiveX è responsabile della connessione delle proprietà associate ai dati che sono state abilitate nella libreria di tipi per il database. Per informazioni su come scrivere un contenitore utilizzando C++Builder, consultare "Uso di controlli ActiveX associati ai dati" a pagina 40-9.

Per attivare l'associazione di dati semplici utilizzare la libreria di tipi:

1 Sulla barra strumenti, fare clic sulla proprietà che si desidera associare.

2 Scegliere la pagina Flags.

3 Scegliere i seguenti attributi di bind:

Attributo	Descrizione
Bindable	Indica che la proprietà supporta l'associazione di dati. Se l'attributo bindable è selezionato, quando il valore della proprietà viene modificato, la proprietà lo notifica al proprio contenitore.
Request Edit	Indica che la proprietà supporta la notifica di OnRequestEdit. Questo consente al controllo di chiedere al contenitore se il valore può essere modificato dall'utente.
Display Bindable	Indica che il contenitore può mostrare agli utenti che la proprietà è bindable.

Attributo	Descrizione
Default Bindable	Indica la singola proprietà bindable che meglio rappresenta l'oggetto. Le proprietà che hanno l'attributo default bindable devono avere anche l'attributo bindable. Non è possibile specificare l'attributo su più di una proprietà di una dispinterface.
Immediate Bindable	Consente alle singole proprietà bindable su una scheda di specificare questo comportamento. Quando questo bit risulta impostato, tutte le modifiche verranno notificate. Affinché questo bit attributo abbia effetto, è necessario impostare i bit di attributo bindable e request edit.

4 Fare clic sul pulsante Refresh sulla barra strumenti per aggiornare la libreria di tipi.

Per poter collaudare un controllo con associazione di dati, è necessario anzitutto registrarlo.

Ad esempio, per convertire un controllo *TEdit* in un controllo ActiveX associato ai dati, si deve creare il controllo ActiveX a partire da *TEdit* e quindi cambiare i flag della proprietà Text in Bindable, Display Bindable, Default Bindable e Immediate Bindable.

Per abilitare la modifica dei dati nel wrapper ActiveX, prima di consentire la modifica, il wrapper deve chiedere al proprio contenitore se è possibile modificare i valori. Per fare ciò, non appena il controllo VCL riceve messaggi derivanti dalla pressione di un tasto da parte dell'utente, chiama il metodo *FireOnRequestEdit*, ereditato dalla classe base *TVclComControl*. Il wrapper ActiveX assegna al controllo VCL un gestore di evento *OnKeyPress*. Il gestore viene modificato in modo che assomigli a quanto segue:

```
void __fastcall TMyAwareEditImpl::KeyPressEvent(TObject *Sender, char &Key)
{
   signed_char TempKey;
   const DISPID dispid = -517; // this is the dispatch id of the data-bound property

   if (FireOnRequestEdit(dispid) == S_FALSE) // ask container if edits ok?
     {
      Key = 0; // if edits not ok, cancel the keypress message
      return;
     }
  TempKey = (signed_char)Key;
  Fire_OnKeyPress(&TempKey);// this forwards the OnKeyPress event to fire in the container
  Key = (signed_char)TempKey;
};
```

Il wrapper ActiveX deve anche notificare al contenitore se i dati hanno subito modifiche. Per fare ciò, non appena i valori del controllo di editing vengono modificati, chiama il metodo *FireOnChanged* (ereditato da *TVclComControl*). *FireOnChanged* notifica al controllo che c'è qualcosa di diverso. Quando i dati nel controllo vengono modificati, il contenitore può mettere il dataset associato in modalità modifica. *FireOnChanged* consente anche la modifica in tempo reale dei campi di dati. Il codice seguente mostra il gestore di evento *OnChange* modificato:

```
void __fastcall TMyAwareEditImpl::ChangeEvent(TObject *Sender)
```

```
{
  const DISPID dispid = -517; // the dispath id of the data-bound property
  FireOnChanged(dispid); // add this line to inform the container of changes
  Fire_OnChange(); // this was the original call to fire the event in the container
};
```

Dopo aver registrato e importato il controllo, è possibile utilizzarlo per visualizzare i dati.

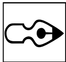
Creazione di una pagina di proprietà per un controllo ActiveX

Una pagina di proprietà è una finestra di dialogo simile all'Object Inspector di C++Builder in cui gli utenti possono cambiare le proprietà di un controllo ActiveX. Una pagina di proprietà consente di raggruppare più proprietà di un controllo, in modo da poterle modificare allo stesso tempo. Oppure, è possibile prevedere una finestra di dialogo per la modifica di proprietà più complesse.

Normalmente, gli utenti accedono alla pagina di proprietà facendo clic con il tasto destro sul controllo ActiveX e scegliendo l'opzione Properties.

Il processo di creazione di una pagina di proprietà è simile a quello della creazione di una scheda.

1 Creare una nuova pagina di proprietà.
2 Aggiungere i controlli alla pagina di proprietà.
3 Associare i controlli sulla pagina di proprietà alle proprietà di un controllo ActiveX.
4 Collegare la pagina di proprietà al controllo ActiveX.



Quando si aggiungono proprietà ad un controllo ActiveX o ad un ActiveForm, è necessario renderle pubbliche se si vuole che siano permanenti. Se non sono pubbliche nel controllo VCL associato, è necessario creare un discendente custom del controllo VCL che dichiari di nuovo le proprietà come pubbliche, utilizzando poi il wizard ActiveX control per creare un controllo ActiveX dalla classe derivata.

Creazione di una nuova pagina di proprietà

Per creare una nuova pagina di proprietà, si utilizza il wizard Property Page.

To create a new property page,

1 Selezionare File | New | Other.
2 Selezionare il separatore ActiveX.
3 Fare doppio clic sull'icona Property Page.

Il wizard crea una nuova scheda e una nuova unit di implementazione per la pagina di proprietà. La scheda è un discendente di *TPropertyPage* il che consente di associare la scheda al controllo ActiveX di cui ne modifica le proprietà. Inoltre, la unit di implementazione dichiara un oggetto implementazione (utilizzando la macro

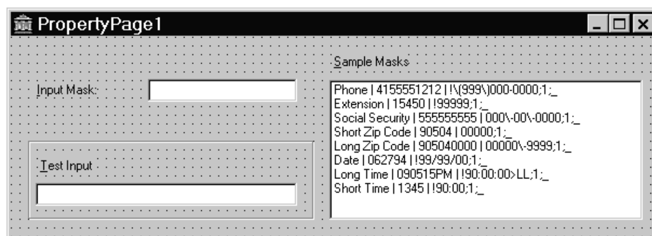
PROPERTYPAGE_IMPL). Questo oggetto implementazione implementa le interfacce della pagina di proprietà, passando alla scheda le opportune chiamate.

Aggiunta di controlli ad una pagina di proprietà

Per ogni proprietà del controllo ActiveX a cui si desidera far accedere l'utente, occorre aggiungere un controllo alla pagina di proprietà.

Per esempio, l'illustrazione seguente mostra una pagina di proprietà per l'impostazione della proprietà MaskEdit di un controllo ActiveX.

Figura 43.1 La pagina di proprietà Mask Edit in modalità progetto



La casella di riepilogo consente all'utente di effettuare una scelta da un elenco di maschere campione. I controlli di modifica consentono all'utente di verificare la maschera prima di applicarla al controllo ActiveX. I controlli vengono aggiunti alla pagina di proprietà in modo analogo all'aggiunta di controlli ad una normale scheda.

Associazione dei controlli di una pagina di proprietà alle proprietà del controllo ActiveX

Dopo aver aggiunto alla pagina di proprietà tutti i controlli necessari, occorre associare ogni controllo alla proprietà corrispondente. Questa associazione viene fatta aggiungendo del codice ai metodi *UpdatePropertyPage* e *UpdateObject* della pagina di proprietà.

Aggiornamento della pagina di proprietà

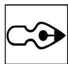
Il metodo *UpdatePropertyPage* viene chiamato per aggiornare i controlli sulla pagina di proprietà quando si cambiano le proprietà del controllo ActiveX. Per aggiornare la pagina di proprietà con i valori correnti delle proprietà del controllo ActiveX, occorre aggiungere del codice al metodo *UpdatePropertyPage*.

È possibile accedere al controllo ActiveX utilizzando la proprietà *OleObject*, che è un *OleVariant* contenente l'interfaccia del controllo ActiveX, della pagina di proprietà.

Per esempio, il codice seguente aggiorna il controllo della pagina di proprietà (InputMask) con il valore corrente della proprietà *EditMask* del controllo ActiveX:

```
void __fastcall TPropertyPage1::UpdatePropertyPage(void)
{
  InputMask->Text = OleObject.OlePropertyGet("EditMask");
```

```
}
```



È anche possibile scrivere una pagina di proprietà che rappresenti più di un controllo ActiveX. In questo caso, non si utilizza la proprietà *OleObject*. Invece, si deve scorrere in sequenza un elenco delle interfacce, che è conservato nella proprietà *OleObjects*.

Aggiornamento dell'oggetto

Il metodo *UpdateObject* viene chiamato quando l'utente apporta dei cambiamenti ai controlli sulla pagina di proprietà. Per impostare le proprietà del controllo ActiveX ai loro nuovi valori, occorre aggiungere del codice al metodo *UpdateObject*.

Ancora una volta si utilizzerà la proprietà *OleObject* per accedere al controllo ActiveX.

Per esempio, il codice seguente imposta la proprietà *EditMask* del controllo ActiveX usando il valore del controllo della casella di testo della pagina di proprietà (InputMask):

```
void __fastcall TPropertyPage1::UpdateObject(void)
{
  // Update OleObject from your control
  OleObject.OlePropertySet<WideString>("EditMask", WideString(InputMast->Text).Copy());
}
```

Collegamento di una pagina di proprietà ad un controllo ActiveX

Per collegare una pagina di proprietà ad un controllo ActiveX:

1 Nella unit di implementazione del controllo ActiveX, all'interno delle istruzione BEGIN_PROPERTY_MAP e END_PROPERTY_MAP, aggiungere una chiamata alla macro PROP_PAGE passandole il GUID della pagina di proprietà. (Il GUID è definito nella unit di implementazione della pagina di proprietà; esso viene generato automaticamente dal wizard Property Page.)

   Ad esempio, se il GUID della pagina di proprietà è definito come CLSID_PropertyPage1 (impostazione predefinita), per un controllo ActiveX basato su una scheda della VCL, la sezione della mappa delle proprietà sarà simile alla seguente:

```
BEGIN_PROPERTY_MAP(TActiveFormXImpl)
   // Define property pages here.  Property pages are defined using
   // the PROP_PAGE macro with the class id of the page.  For example,
   // PROP_PAGE(CLSID_ActiveFormXPage)
   PROP_PAGE(CLSID_PropertyPage1)
END_PROPERTY_MAP()
```

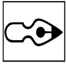
2 Includere la unit della pagina di proprietà nella unit del controllo ActiveX.

Registrazione di un controllo ActiveX

Dopo aver creato il controllo ActiveX, occorre registrarlo in modo che le altre applicazioni possano trovarlo ed utilizzarlo.

Per registrare un controllo ActiveX:

- Scegliere Run | Register ActiveX Server.



Prima di rimuovere un controllo ActiveX dal sistema, occorrerà annullarne la registrazione.

Per annullare la registrazione di un controllo ActiveX:

- Scegliere Run | Unregister ActiveX Server.

Come metodo alternativo, è possibile utilizzare il comando tregsvr dalla riga comandi oppure eseguire il programma regsvr32.exe dal sistema operativo.

Collaudo di un controllo ActiveX

Per verificare il controllo, aggiungerlo ad un package ed importarlo come controllo ActiveX. Questa procedura aggiunge il controllo ActiveX alla Component palette di C++Builder. È possibile trascinare il controllo su una scheda ed effettuare tutte le verifiche necessarie.

Il controllo dovrebbe essere verificato anche in tutte le applicazioni di destinazione che lo useranno.

Per effettuare il debug del controllo ActiveX, selezionare Run | Parameters e scrivere il nome del client nella casella di testo Host Application.

I parametri vengono, quindi, applicati all'applicazione host. Selezionando Run | Run verrà eseguita l'applicazione host o client e si potranno impostare i breakpoint nel controllo.

Distribuzione di un controllo ActiveX su Web

Prima che i controlli ActiveX creati possano essere usati dai client Web, essi devono essere distribuiti sul server Web. Ogni volta che si apportano modifiche ad un controllo ActiveX, è necessario ricompilarlo e ridistribuirlo in modo che le applicazioni client possano vedere le modifiche.

Prima di poter distribuire il controllo ActiveX, occorre avere un server per Web che risponda ai messaggi dei client.

Per distribuire un controllo ActiveX:

1. Selezionare Project | Web Deployment Options.
2. Sulla pagina Project impostare Target Dir alla posizione della DLL del controllo ActiveX come percorso sul server Web. Questo può essere un nome di percorso locale o un percorso UNC, per esempio, C:\INETPUB\wwwroot.
3. Impostare Target URL alla posizione della DLL del controllo ActiveX, espressa come Uniform Resource Locators (URL) e senza indicare il nome del file, sul server Web, per esempio, http://mymachine.inprise.com/. Per maggiori

informazioni su come eseguire queste operazioni, consultare la documentazione del server Web.

4 Impostare HTML Dir alla posizione (espressa come un percorso) in cui dovrebbe essere collocato il file HTML che contiene un riferimento al controllo ActiveX, per esempio, C:\INETPUB\wwwroot. Questo percorso può essere un nome di percorso standard o un percorso UNC.

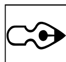
5 Impostare le opzioni desiderate per la distribuzione su Web come descritto nella sezione “Impostazione delle opzioni” a pagina 43-18.

6 Scegliere OK.

7 Scegliere Project | Web Deploy.

   In questo modo viene creata una base per la distribuzione che contiene il controllo ActiveX in una libreria ActiveX (con estensione OCX). In base alle opzioni specificate, questa base di distribuzione può contenere anche un file cabinet (con estensione CAB) o un file di informazioni (con estensione INF).

   La libreria ActiveX viene collocata nella Target Directory indicata al punto 2. Il file HTML ha lo stesso nome del file di progetto ma con estensione HTM. Viene creato nella HTML Directory indicata al punto 4. Il file HTML contiene un riferimento URL alla libreria ActiveX presente alla posizione specificata al punto 3.



   Se si desidera collocare questi file sul server Web, utilizzare un programma di utilità esterno come ftp.

8 Attivare il browser Web che supporta ActiveX e visualizzare la pagina HTML creata.

Quando questa pagina HTML viene visualizzata nel browser Web, la scheda o il controllo vengono visualizzati ed eseguiti come un’applicazione incorporata all’interno del browser. Vale a dire, la libreria viene eseguita nello stesso processo dell’applicazione browser.

Impostazione delle opzioni

Prima di distribuire un controllo ActiveX, occorre specificare le opzioni di distribuzione su Web che si dovrebbero seguire durante la creazione della libreria ActiveX.

Le opzioni di distribuzione su Web consentono di impostare i seguenti parametri:

- **Including additional files:** Se il controllo ActiveX dipende da altri package o da altri file aggiuntivi, è possibile indicare che questi dovrebbero essere distribuiti col progetto. Per impostazione predefinita, questi file utilizzano le stesse opzioni specificate per l’intero progetto, ma è possibile ridefinire queste impostazioni utilizzando le pagine Package o Additional files. Quando si includono package o file aggiuntivi, C++Builder crea un file con estensione .INF (che sta per INFormazioni). Questo file specifica i vari file che devono essere scaricati e configurati affinché sia possibile eseguire la libreria ActiveX. La sintassi del file INF consente di scaricare URL che puntino a package o a file aggiuntivi.

- **CAB file compression**: Un cabinet è singolo file, di solito con estensione **CAB**, che memorizza i file compressi in una libreria di file. La compressione cabinet può diminuire significativamente (fino al 70%) il tempo di prelievo di un file. Durante l'installazione il browser espande i file memorizzati in un cabinet e li copia nel sistema dell'utente. Ogni file distribuito può essere un file CAB compresso. Sulla pagina Project della finestra di dialogo Web Deployment options è possibile specificare che la libreria ActiveX utilizza file CAB compressi.
- **Version information**: È possibile specificare che si desidera includere nel controllo ActiveX le informazioni sulla versione. Queste informazioni vengono impostate nella pagina VersionInfo della finestra di dialogo Project Options. Fra queste informazioni c'è il numero di versione che è possibile far aggiornare automaticamente ogni volta che si distribuisce il controllo ActiveX. Se si includono package o file aggiuntivi, anche le rispettive risorse di informazioni di Versione potrebbero essere aggiunte al file di INF.

A seconda del fatto che si includano file aggiuntivi e/o si utilizzino file CAB compressi, la libreria ActiveX risultante può essere un file OCX, un file CAB contenente un file OCX, o un file INF. La tabella seguente riassume i risultati derivanti dalla scelta di varie combinazioni.

Package e/o file aggiuntivi	Compressione di file CAB	Risultato
No	No	Un file di libreria ActiveX (OCX.
No	Sì	Un file CAB contenente un file di libreria ActiveX.
Sì	No	Un file .INF, un file di libreria ActiveX ed eventuali altri file e package.
Sì	Sì	Un file INF, un file CAB contenente un file di libreria ActiveX, e un file CAB per ciascun file o package aggiuntivo.

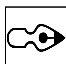
Capitolo

44

Creazione di oggetti MTS o COM+

C++Builder utilizza il termine "oggetti transazionali" per fare riferimento a oggetti che sfruttano i servizi di transazione, di sicurezza e di gestione delle risorse forniti da Microsoft Transaction Server (MTS) (per le versioni di Windows antecedenti a Windows 2000) o da COM+ (per Windows 2000 e versioni successive). Questi oggetti sono progettati per operare in vasti ambienti distribuiti. Non possono essere utilizzati in applicazioni multipiattaforma a causa della loro dipendenza dalla tecnologia specifica di Windows.

C++Builder dispone di un wizard che crea oggetti transazionali così che possono sfruttare i vantaggi degli attributi COM+ o dell'ambiente MTS. Queste caratteristiche forniscono molti servizi accessori destinati a semplificare la creazione e l'implementazione di client e server COM, in particolare di server remoti.



Per le applicazioni database, C++Builder fornisce anche un Transactional Data Module. Per ulteriori informazioni, consultare il Capitolo 29, "Creazione di applicazioni multi-tier".

Gli oggetti transazionali utilizzano diversi servizi a basso livello, come:

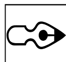
- La gestione delle risorse di sistema, compresi i processi, i thread e le connessioni al database in modo che l'applicazione server possa gestire molti utenti simultaneamente.
- Iniziazione e controllo automatico delle transazioni in modo che l'applicazione sia affidabile.
- Creazione, esecuzione e cancellazione, quando necessario, dei componenti server.
- Fornitura di servizi di sicurezza in base al ruolo, in modo che solo gli utenti autorizzati possano accedere all'applicazione.
- Gestione degli eventi in modo che i client possano rispondere alle condizioni che si presentano sul server (solo per COM+).

Lasciando a MTS o a COM+ il compito di provvedere ai servizi di base, è possibile concentrarsi sullo sviluppo dei particolari della specifica applicazione distribuita. La tecnologia scelta (MTS o COM+) dipende dal server su cui si decide di far eseguire l'applicazione. Dal punto di vista dei client, la differenza tra le due tecnologie (o, per quel che ci riguarda, il fatto che l'oggetto server utilizzi qualcuno di questi servizi) è trasparente (a meno che il client intervenga esplicitamente sui servizi transazionali mediante una speciale interfaccia).

Gli oggetti transazionali

Normalmente, gli oggetti transazionali sono piccoli, e vengono usati per funzioni di gestione discrete. Essi possono implementare le regole di gestione di un'applicazione, fornendo le relative viste e le trasformazioni di stato. Si consideri, per esempio, il caso di un'applicazione clinica di un medico. I record clinici memorizzati in diversi database rappresentano lo stato persistente dell'applicazione, come la storia clinica di un paziente. Gli oggetti transazionali aggiornano quello stato in modo da riflettere le modifiche quali, ad esempio, i nuovi pazienti, i risultati delle analisi del sangue e le radiografie.

Gli oggetti transazionali si distinguono dagli altri oggetti COM per il fatto che utilizzano un set di attributi forniti da MTS o COM+ per la gestione delle problematiche che possono sorgere in un ambiente elaborativo distribuito. Alcuni di questi attributi richiedono che l'oggetto transazionale implementi l'interfaccia *IObjectControl*. *IObjectControl* definisce i metodi che vengono chiamati al momento dell'attivazione o della disattivazione dell'oggetto, in cui è possibile gestire risorse come le connessioni ai database. Tale interfaccia è necessaria anche per la condivisione degli oggetti, descritta in "Condivisione degli oggetti" a pagina 44-10.



Se si utilizza MTS, gli oggetti transazionali devono implementare *IObjectControl*. In ambiente COM+, *IObjectControl* non è necessario ma se ne raccomanda l'utilizzo. Il Transactional Object wizard fornisce un oggetto che deriva da *IObjectControl*.

Un client di un oggetto transazionale è detto client base. Dal punto di vista di un client base, un oggetto transazionale assomiglia ad un qualsiasi altro oggetto COM.

In ambiente MTS, l' oggetto transazionale deve essere incluso in una libreria (DLL) che deve essere poi installata nell'ambiente di esecuzione di MTS (l'eseguibile di MTS, mtxex.exe). Ovvero, l'oggetto server viene eseguito nello spazio del processo di esecuzione di MTS. L'eseguibile di MTS può essere in esecuzione nello stesso del client base, come processo separato sulla stessa macchina del client base o come processo di server remoto su una macchina separata.

In ambiente COM+, non è necessario che l'applicazione server sia un server in-process. Poiché i vari servizi sono integrati nelle librerie COM, non c'è necessità che esista un processo MTS separato per intercettare le chiamate al server. Invece, è COM stesso (o meglio, COM+) a fornire la gestione delle risorse, il supporto per le transazioni e così via. Però, l'applicazione server deve ancora essere installata questa volta in un'applicazione di COM+.

La connessione tra il client base e l'oggetto transazionale è gestita da un proxy sul client e da un stub sul server, esattamente come con qualsiasi server out-of-process.

Le informazioni della connessione sono conservate dal proxy.La connessione tra il client base e il proxy rimane aperta finché il client non richiede una connessione al server, facendo così sembrare che il client abbia continuamente avuto accesso al server. Nella realtà, invece, il proxy potrebbe disattivare e riattivare l'oggetto, risparmiando così risorse in modo da consentire ad altri client di utilizzare la connessione. Per maggiori informazioni sull'attivazione e la disattivazione, consultare "Attivazione just-in-time" a pagina 44-5.

Requisiti di un oggetto transazionale

Oltre ai requisiti COM, un oggetto transazionale deve soddisfare i seguenti requisiti:

- L'oggetto deve avere una fabbrica di classi standard. Questa viene preparata automaticamente dal wizard all'atto della creazione dell'oggetto.
- Il server deve esporre i propri oggetti classe esportando il metodo *DllGetClassObject*. Il codice per questa operazione è preparato dal wizard.
- Tutte le interfacce degli oggetti e le CoClass devono essere descritte da una libreria di tipi, che viene creata automaticamente dal wizard. È possibile aggiungere metodi e proprietà alle interfacce nella libreria di tipi usando il Type Library Editor. Le informazioni contenute in questa libreria sono utilizzate da MTS Explorer o da COM+ Component Manager per estrarre in fase di esecuzione le informazioni sui componenti installati.
- Il server deve esportare solo le interfacce che usano lo smistamento standard COM. Ciò viene fatto automaticamente dal Transactional Object wizard. Il supporto di C++Builder per gli oggetti transazionali non consente lo smistamento manuale delle interfacce custom. Tutte le interfacce devono essere implementate come interfacce duali che usano il supporto automatico dello smistamento di COM.
- Il server deve esportare la funzione *DllRegisterServer* e in questa routine eseguire la registrazione automatica del CLSID, del ProgID, delle interfacce e della libreria di tipi. Tutto ciò viene fornito da Transactional Object wizard.

Se si utilizza MTS al posto di COM+, valgono anche le seguenti condizioni:

- MTS richiede che il server sia una libreria a collegamento dinamico (DLL). I server implementati come file eseguibili (file .EXE) non possono essere eseguiti in ambiente MTS.
- L'oggetto deve implementare l'interfaccia *IObjectControl*. Il supporto per questa interfaccia viene aggiunto automaticamente dal Transactional Object wizard.
- Un server che viene eseguito nello spazio di processo MTS non può aggregarsi con oggetti COM che non sono eseguiti in MTS.

Gestione delle risorse

Gli oggetti transazionali possono essere amministrati per gestire meglio le risorse utilizzate dall'applicazione. Queste risorse includono tutto, dalla memoria per le

istanze degli oggetti a qualsiasi risorsa da loro utilizzata (come le connessioni ai database).

In generale, l'applicazione viene configurata per come gestire le risorse in base al modo in cui si installa e configura l'oggetto. Si imposta l'oggetto transazionale in modo che possa sfruttare:

- Attivazione just-in-time
- Condivisione delle risorse
- Condivisione degli oggetti (solo per COM+)

Tuttavia, se si vuole che l'oggetto sfrutti appieno questi servizi, si deve utilizzare l'interfaccia *IObjectContext* per indicare quando è possibile rilasciare le risorse in tutta sicurezza.

Accesso al contesto dell'oggetto

Come con qualsiasi oggetto COM, prima di poter utilizzare un oggetto transazionale è necessario crearlo. I client COM creano un oggetto chiamando la funzione *CoCreateInstance* della libreria COM.

Ogni oggetto transazionale deve avere un oggetto contesto corrispondente. Questo oggetto contesto è implementato automaticamente da MTS o da COM+ ed è utilizzato per gestire l'oggetto transazionale. L'interfaccia dell'oggetto contesto è *IObjectContext*. Gli oggetti transazionali creati automaticamente dal Transactional Object wizard recuperano l'oggetto contesto al momento dell'attivazione. L'oggetto contesto viene memorizzato in una variabile membro di nome m_spObjectContext. Per esempio, è possibile utilizzare il puntatore all'oggetto contesto come segue:

```
BOOL flags;
m_spObjectContext->IsCallerInRole(OLESTR("Manager"), &flags);
```

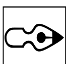
È possibile ottenere un puntatore all'oggetto contesto anche chiamando il metodo *Get_ObjectContext* di *TMtsDll*. L'oggetto *TMtsDll* adatta la chiamata per recuperare l'oggetto contesto, in modo da funzionare indipendentemente dal fatto che l'applicazione sia eseguita in ambiente MTS o COM+:

```
IObjectContext* IAmWatchingYou = NULL;
TMtsDll MTSDLL;
HRESULT hr = MTSDLL.Get_ObjectContext(&IAmWatchingYou);
if (! (SUCCEEDED(hr)) )
{
  // do something useful with the error
}

BOOL flags;
IAmWatchingYou->IsCallerInRole(OLESTR("Manager"), &flags);
```



Utilizzare il metodo *Get_ObjectContext* di *TMtsDll* al posto della macro *GetObjectContext* definita in comsvcs.h. Quest'ultima non è definita negli header della VCL per evitare la ridefinizione del metodo *GetObjectContext* di *TDatabase*.



Il membro m_spObjectContext viene assegnato utilizzando il metodo *TMtsDll*, che funziona sia in ambiente MTS sia in ambiente COM+.

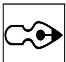
Attivazione just-in-time

La possibilità per un oggetto di essere disattivato e riattivato mentre i client contengono riferimenti ad esso, viene chiamata **attivazione just-in-time**. Dal punto di vista del client, solo una singola istanza dell'oggetto esiste dal momento in cui il client lo crea al momento in cui lo rilascia definitivamente. In realtà, è possibile che l'oggetto sia stato disattivato e riattivato molte volte. Avendo disattivato gli oggetti, i client possono contenere riferimenti all'oggetto per un tempo illimitato senza influenzare le risorse di sistema. Quando un oggetto viene disattivato, ne rilascia tutte le risorse. Per esempio, se si disattiva un oggetto, potrebbe rilasciare la sua connessione al database, consentendo così ad altri oggetti di utilizzarlo.

Un oggetto transazionale viene creato in un stato disattivato e diviene attivo alla ricezione di una richiesta del client. Quando si crea un oggetto transazionale, viene anche creato il corrispondente oggetto contesto. Questo oggetto contesto esiste per tutta la durata dell'oggetto transazionale, attraverso uno o più cicli di riattivazione. L'oggetto contesto, a cui si accede tramite l'interfaccia *IObjectContext*, tiene traccio dell'oggetto durante la disattivazione e coordina le transazioni.

Gli oggetti transazionali vengono disattivati non appena è possibile farlo in tutta sicurezza. Questo comportamento è chiamato **disattivazione as-soon-as-possible**. Un oggetto transazionale viene disattivato quando si verifica una delle seguenti condizioni:

- **L'oggetto richiede la disattivazione con *SetComplete* o *SetAbort***: Un oggetto chiama il metodo *IObjectContext SetComplete* quando ha completato con successo il suo lavoro e non occorre salvare il suo stato interno per la successiva chiamata da parte del client. Un oggetto chiama *SetAbort* per indicare che non può completare con successo il suo lavoro e che il suo stato non ha bisogno di essere salvato. Cioè, lo stato dell'oggetto ritorna alla situazione in cui si trovava prima della transazione corrente. Spesso gli oggetti possono essere progettati per essere senza stato, questo vuol dire che vengono disattivati al ritorno da ogni metodo.
- **Una transazione viene effettuata o abbandonata**: Quando la transazione di un oggetto viene effettuata o abbandonata, l'oggetto viene disattivato. Di questi oggetti disattivati, gli unici che continuano ad esistere sono quelli che sono oggetto di riferimento da parte dei client all'esterno della transazione. Le successive chiamate a questi oggetti li riattivano, consentendo la loro esecuzione nella nuova transazione.
- **L'ultimo client rilascia l'oggetto**: Naturalmente, quando un client rilascia l'oggetto, questo viene disattivato, ed anche il contesto dell'oggetto viene rilasciato.



Se si installa l'oggetto transazionale in ambiente COM+ dall'IDE, è possibile specificare se l'oggetto supporta l'attivazione just-in-time utilizzando la pagina COM+ del Type Library editor. È sufficiente selezionare l'oggetto (CoClass) nel Type Library editor, andare alla pagina COM+, e selezionare o deselezionare la casella Just In Time Activation. Altrimenti, un amministratore di sistema deve specificare questo attributo utilizzando COM+ Component Manager o MTS Explorer.
(L'amministratore di sistema può anche ridefinire qualsiasi impostazione specifica utilizzando il Type Library editor.)

Condivisione delle risorse

Dal momento che le risorse inattive di sistema vengono liberate durante la disattivazione, le risorse liberate sono disponibili per gli altri oggetti server. Per esempio, una connessione al database che non è più utilizzata da un oggetto server può essere riutilizzata da un altro client. Questa tecnica viene chiamata **condivisione delle risorse**. Le risorse condivise sono gestite da un distributore di risorse.

Un distributore di risorse mette in cache le risorse, in modo che tutti gli oggetti transazionali installati possano condividerle. Il distributore di risorse gestisce anche le informazioni di stato non permanenti condivise. In questo senso, i distributori di risorse sono simili ai manager di risorse degli SQL Server, ma senza la garanzia di durabilità.

Quando si scrive un oggetto transazionale, è possibile sfruttare i due tipi di distributori di risorse già forniti:

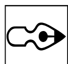
- Distributori di risorse di database
- Shared Property manager

Prima che altri oggetti possano utilizzare le risorse condivise, è necessario rilasciarle esplicitamente.

Distributori di risorse di database

L'apertura e la chiusura di connessioni a un database possono utilizzare molto tempo. Se si utilizza un distributore di risorse per raccogliere le connessioni al database, l'oggetto può riutilizzare le connessioni al database esistenti invece di crearne di nuove. Per esempio, se in un'applicazione di manutenzione del cliente sono attivi un componente per la consultazione del database e un componente per l'aggiornamento del database, è possibile installare insieme quei componenti, in modo che possano condividere le connessioni al database. In questo modo l'applicazione non avrà bisogno di molte connessioni e le nuove istanze degli oggetto potranno accedere più rapidamente ai dati utilizzando una connessione già aperta ma non utilizzata.

- Se si stanno utilizzando componenti BDE per connettersi ai dati, il distributore di risorse è Borland Database Engine (BDE). Questo distributore di risorse è disponibile solo quando l'oggetto transazionale è installato con MTS. Per attivare il distributore di risorse, utilizzare BDE Administrator per impostare a ON il parametro MTS POOLING nell'area di configurazione System/Init.
- Se si stano utilizzando componenti database di ADO per connettersi ai dati, il distributore di risorse è fornito da ADO.



Se si stanno utilizzando componenti InterbaseExpress per l'accesso al database, non esiste alcun sistema precostituito per la condivisione delle risorse.

Per i moduli dati transazionali remoti, le connessioni vengono automaticamente impostate in base alle transazioni di un oggetto, e il distributore di risorse può ricuperare e riutilizzare automaticamente le connessioni.

Shared property manager

Lo Shared Property Manager è un distributore di risorse che è possibile utilizzare per condividere lo stato tra più oggetti all'interno di un processo del server. Utilizzando Shared Property manager, si evita l'aggiunta all'applicazione di molto codice per gestire i dati condivisi: Shared Property manager li gestisce automaticamente implementando blocchi e semafori in modo da proteggere le proprietà condivise da accessi simultanei. Shared Property manager elimina le collisioni sui nomi mettendo a disposizione **gruppi di proprietà condivise** che stabiliscono spazi di nome univoci per le proprietà condivise in essi contenute.

Per utilizzare la risorsa dello Shared Property manager, si deve per prima cosa usare la funzione helper *CreateSharedPropertyGroup* per creare un gruppo di proprietà condivise. Poi si possono scrivere tutte le proprietà in quel gruppo e leggerle da quel gruppo. Utilizzando un gruppo di proprietà condivise, le informazioni di stato vengono salvate tutte le volte che si disattiva un oggetto transazionale. Inoltre, le informazioni di stato possono essere condivise tra tutti gli oggetti transazionali installati nello stesso package MTS o nella stessa applicazione COM+. È possibile installare oggetti transazionali in un package secondo la modalità descritta nel paragrafo "Installazione di oggetti transazionali" a pagina 44-29.

Affinché gli oggetti condividano lo stato, devono essere eseguiti tutti nello stesso processo. Se si desidera che istanze di componenti diversi condividano proprietà, è necessario installarli nello stesso package MTS o nella stessa applicazione COM+. Poiché c'è il rischio che gli amministratori spostino componenti da un package a un altro, è più sicuro limitare l'uso di un gruppo di proprietà condivise a istanze di oggetti definiti nella stessa DLL o nello stesso EXE.

Gli oggetti che condividono proprietà devono avere lo stesso attributo di attivazione. Se due componenti nello stesso package hanno attributi di attivazione differenti, normalmente non potranno condividere le proprietà. Per esempio, se un componente è configurato per funzionare nel processo di un client e l'altro è configurato per funzionare in un processo del server, i loro oggetti di solito saranno eseguiti in processi diversi, anche se sono inclusi nello stesso package MTS o nella stessa applicazione COM+.

L'esempio seguente mostra come aggiungere del codice per supportare lo Shared Property manager in un oggetto transazionale:

Esempio: Condivisione di proprietà tra istanze di un oggetto transazionale

Questo esempio crea un gruppo di proprietà di nome MyGroup per contenere le proprietà da condividere tra oggetti e istanze di oggetti. In questo esempio c'è una proprietà Counter che viene condivisa. L'esempio utilizza la funzione helper *CreateSharedPropertyGroup* per creare il manager del gruppo di proprietà e il gruppo di proprietà, e poi utilizza il metodo *CreateProperty* dell'oggetto Group per creare una proprietà di nome Counter.

Per ottenere il valore di una proprietà, si utilizza il metodo *PropertyByName* dell'oggetto Group, come mostrato di seguito. È anche possibile utilizzare il metodo *PropertyByPosition*.

```
#include "Project1_TLB.H"
#define _MTX_NOFORCE_LIBS
```

```
#include <vcl\mtshlpr.h>

class ATL_NO_VTABLE TSharedPropertyExampleImpl :
  public CComObjectRootEx<CComSingleThreadModel>,
  public CComCoClass<TSharedPropertyExampleImpl, &CLSID_SharedPropertyExample>,
  public IObjectControl,
  public IDispatchImpl<ISharedPropertyExample, &IID_ISharedPropertyExample, &LIBID_Project1>
{
private:
  ISharedPropertyGroupManager* manager;
  ISharedPropertyGroup* PG13;
  ISharedProperty* Counter;
public:
  TSharedPropertyExampleImpl()
  {
  }
  DECLARE_THREADING_MODEL(otApartment);
  DECLARE_PROGID("Project1.SharedPropertyExample");
  DECLARE_DESCRIPTION("");

static HRESULT WINAPI UpdateRegistry(BOOL bRegister)
{
  TTypedComServerRegistrarT<TSharedPropertyExampleImpl>
  regObj(GetObjectCLSID(), GetProgID(), GetDescription());
  return regObj.UpdateRegistry(bRegister);
}

DECLARE_NOT_AGGREGATABLE(TSharedPropertyExampleImpl)

BEGIN_COM_MAP(TSharedPropertyExampleImpl)
  COM_INTERFACE_ENTRY(ISharedPropertyExample)
  COM_INTERFACE_ENTRY(IObjectControl)
  COM_INTERFACE_ENTRY(IDispatch)
END_COM_MAP()

public:
  STDMETHOD(Activate)();
  STDMETHOD(IncrementCounter());
  STDMETHOD_(BOOL, CanBePooled)();
  STDMETHOD_(void, Deactivate)();

CComPtr<IObjectContext> m_spObjectContext;

public:

};

// SHAREDPROPERTYEXAMPLEIMPL : Implementation of TSharedPropertyExampleImpl

#include <vcl.h>
#pragma hdrstop

#include "SHAREDPROPERTYEXAMPLEIMPL.H"

STDMETHODIMP TSharedPropertyExampleImpl::Activate()
{
  static TMtsDll Mts;
  HRESULT hr = E_FAIL;
  hr = Mts.Get_ObjectContext(&m_spObjectContext);
  if (SUCCEEDED(hr))
```

```
    {
      VARIANT_BOOL _false = VARIANT_FALSE;
      VARIANT_BOOL _true = VARIANT_TRUE;
      CoCreateInstance( CLSID_SharedPropertyGroupManager, NULL, CLSCTX_INPROC_SERVER,
                        IID_ISharedPropertyGroupManager, (void**)manager);
      manager->CreatePropertyGroup(L"Test Group", LockSetGet, Standard, &_false, &PG13);
      if ( (PG13->CreateProperty(L"Counter", &_true, &Counter)) == S_OK)
      {
        Counter->put_Value(TVariant(0));
      }
      return S_OK;
    }
  return hr;
}
STDMETHODIMP_(BOOL) TSharedPropertyExampleImpl::CanBePooled()
{
  return FALSE;
}
STDMETHODIMP_(void) TSharedPropertyExampleImpl::Deactivate()
{
  PG13->Release();
  manager->Release();
  m_spObjectContext.Release();
}
STDMETHODIMP TSharedPropertyExampleImpl::IncrementCounter()
{
  try
  {
    TVariant temp;
    Counter->get_Value(&temp);
    temp=(int)temp+1;;
    Counter->put_Value(temp);
  }
  catch(Exception &e)
  {
    return Error(e.Message.c_str(), IID_ISharedPropertyExample);
  }
  return S_OK;
};
```

Rilascio delle risorse

Il programmatore si deve far carico di rilasciare le risorse di un oggetto. Di solito, questa operazione viene effettuata chiamando i metodi *IObjectContext*, *SetComplete* e *SetAbort* dopo aver soddisfatto le richieste di servizio dei client. Questi metodi rilasciano le risorse allocate dal distributore di risorse.

Contemporaneamente occorre rilasciare i riferimenti a tutte le altre risorse, inclusi i riferimenti ad altri oggetti (compresi gli oggetti transazionali e gli oggetti contestuali) e la memoria occupata da tutte le istanze del componente (liberando il componente).

L'unico caso in cui non è necessario includere queste chiamate è quando si desidera conservare lo stato tra le chiamate del client. Per informazioni dettagliate, consultare "Oggetti con e senza stato" a pagina 44-13.

Condivisione degli oggetti

Esattamente come è possibile costituire un fondo di risorse, in ambiente COM+ è possibile costituire un fondo di oggetti. Al momento della disattivazione di un oggetto, COM+ chiama il metodo *CanBePooled* dell'interfaccia *IObjectControl*, il quale indica se l'oggetto può essere condiviso per il riutilizzo. Se *CanBePooled* restituisce true, anziché essere distrutto al momento della disattivazione, l'oggetto viene spostato fra quelli che possono essere condivisi. Esso rimane nel fondo comune degli oggetti per un periodo di tempo specificato, durante il quale resta disponibile per l'utilizzo da parte di un qualsiasi client che ne faccia richiesta. Solo quando il fondo comune di oggetti è vuoto viene creata una nuova istanza dell'oggetto. Gli oggetti che restituiscono false o che non supportano l'interfaccia *IObjectControl* al momento della disattivazione vengono distrutti.

La condivisione degli oggetti non è disponibile in ambiente MTS. MTS chiama *CanBePooled* come descritto, ma non ha luogo alcuna condivisione. A causa di ciò, quando il wizard crea gli oggetti transazionali, il metodo *CanBePooled* generato restituisce sempre **false**. Se l'oggetto sarà eseguito solamente in ambiente COM+ e si vuole consentire la costituzione di un fondo di oggetti, si deve individuare questo metodo nella unit di implementazione e modificarlo in modo che restituisca **true**.

Anche se il metodo *CanBePooled* di un oggetto restituisce **true**, può essere configurato in odo che COM+ non sposti nel fondo comune degli oggetti. Se si installa l'oggetto transazionale in ambiente COM+ dall'IDE, utilizzando la pagina COM+ del Type Library editor, è possibile specificare se COM+ può tentare di includere l'oggetto nel fondo comune. È sufficiente selezionare l'oggetto (la CoClass) nell'editor della libreria di tipi, andare alla pagina COM+, e selezionare o deselezionare la casella Object Pooling. Altrimenti, un amministratore di sistema deve specificare questo attributo utilizzando COM+ Component Manager o MTS Explorer.

Analogamente, è possibile configurare per quanto tempo un oggetto disattivato può restare nel fondo comune degli oggetti prima di essere liberato. Se lo si sta installando dall'IDE, è possibile specificare questa durata utilizzando l'impostazione Creation TimeOut disponibile sulla pagina COM+ dell'editor della libreria di tipi. In alternativa, questo attributo può essere specificato da un amministratore di sistema utilizzando COM+ Component Manager o MTS Explorer.

Supporto delle transazioni MTS e COM+

Il supporto delle transazioni, che dà il nome agli oggetti transazionali, consente di raggruppare azioni in transazioni. Per esempio, in un'applicazione di record clinici, se ci fosse un componente Transfer per trasferire i record da un medico all'altro, si potrebbero includere i metodi Add e Delete nella stessa transazione. In questo modo, o il trasferimento avverrebbe senza problemi oppure il tutto verrebbe riportato allo

stato precedente. Le transazioni semplificano il recupero degli errori per le applicazioni che devono accedere *a più* database.

Le transazioni assicurano che:

- Tutti gli aggiornamenti di una singola transazione vengono effettuati oppure vengono abbandonati e riportati al loro stato precedente. Questa operazione viene detta **atomicità**.
- Una transazione è una trasformazione corretta dello stato di un sistema, che preserva gli invarianti di stato. Questa operazione viene detta **consistenza**.
- Le transazioni concorrenti non vedono i rispettivi risultati parziali e non completati, il che potrebbe creare incompatibilità nello stato dell'applicazione. Questa operazione viene detta **isolamento**. I gestori delle risorse usano protocolli di sincronizzazione basati sulle transazioni per isolare il lavoro non completato delle transazioni attive.
- Gli aggiornamenti effettuati alle risorse gestite (come i record di un database) sopravvivono ai guasti, compresi i problemi di comunicazione, di processo e del sistema server. Questa operazione viene detta **durabilità**. La registrazione transazionale consente di recuperare lo stato durevole dopo un guasto dei supporti di memorizzazione su disco.

Un oggetto contesto associato ad un oggetto indica se l'oggetto è eseguito all'interno di una transazione e, in tal caso, l'identità della transazione. Quando un oggetto fa parte di una transazione, i servizi che i manager di risorse e i distributori di risorse compiono per suo conto vengono eseguite anch'essi nella stessa transazione. I distributori di risorse utilizzano l'oggetto contesto per fornire servizi basati sulla transazione. Per esempio, quando un oggetto, eseguito all'interno di una transazione, alloca una connessione al database utilizzando il distributore di risorse ADO o BDE, la connessione è assimilata automaticamente nella transazione. Tutti gli aggiornamenti al database che utilizzano questa connessione diventano parte della transazione, e vengono o confermati o annullati.

Le operazioni derivanti da più oggetti possono essere composti in una singola transazione. La possibilità per un oggetto di esistere nella propria transazione o di essere parte di un più vasto gruppo di oggetti, appartenenti a una singola transazione, è uno dei maggiori vantaggi offerti da MTS e COM+. Essa consente di utilizzare un oggetto in vari modi, tanto che chi sviluppa applicazioni può riutilizzare il codice dell'applicazione in applicazioni diverse senza riscrivere la logica dell'applicazione. Infatti, gli sviluppatori possono determinare come saranno utilizzati gli oggetti nelle transazioni al momento dell'installazione dell'oggetto transazionale. Essi possono modificare il comportamento della transazione semplicemente aggiungendo un oggetto a un package MTS o ad un'applicazione COM+ differente. Per maggiori informazioni sull'installazione di oggetti transazionali, consultare il paragrafo "Installazione di oggetti transazionali" a pagina 44-29.

Attributi di transazione

Ciascun oggetto transazionale ha un attributo di transazione che viene registrato nel catalogo MTS o memorizzato con COM+.

C++Builder consente di impostare l'attributo di transazione in fase di progetto utilizzando il wizard dell'oggetto transazionale o il Type Library editor.

Ciascun attributo di transazione può essere impostato con questi parametri:

Requires a transaction	Gli oggetti devono essere eseguiti *all'interno del campo d'azione di una transazione*. Quando viene creato un nuovo oggetto, il suo contesto eredita la transazione dal contesto del client. Se il client non ha un contesto di transazione, ne crea automaticamente uno nuovo.
Requires a new transaction	Gli oggetti devono essere eseguiti *all'interno delle loro transazioni*. Quando viene creato un nuovo oggetto, una nuova transazione viene creata automaticamente per l'oggetto, indipendentemente dal fatto che il suo client abbia una transazione. Un oggetto non viene mai eseguito all'interno del campo d'azione della transazione del suo client. Il sistema, invece, crea sempre transazioni indipendenti per i nuovi oggetti.
Supports transactions	Gli oggetti possono essere eseguiti *all'interno del campo d'azione delle transazioni del loro client*. Quando viene creato un nuovo oggetto, il suo contesto eredita la transazione dal contesto del client. Questo attributo rende possibile la composizione di più oggetti in un'unica transazione. Se il client non ha una transazione, anche il nuovo contesto viene creato senza.
Transactions Ignored	Gli oggetti *non vengono eseguiti nel campo d'azione delle transazioni*. Quando si crea un nuovo oggetto, il suo contesto di oggetto viene creato senza una transazione, indipendentemente da fatto che il client abbia una transazione. Questa impostazione è disponibile solamente in ambiente COM+.
Does not support transactions	Il significato di questa impostazione varia, in base al fatto che si stia installando l'oggetto in ambiente MTS o COM+. In ambiente MTS, questa impostazione ha lo stesso significato di Transactions Ignored in ambiente COM+. In ambiente COM+, non solo il contesto dell'oggetto viene creato senza una transazione, ma questa impostazione impedisce di attivare l'oggetto se il client ha una transazione.

Impostazione dell'attributo di transazione

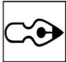
È possibile impostare un attributo di transazione al momento della creazione di un oggetto transazionale utilizzando Transactional Object wizard.

È anche possibile impostare (o modificare) l'attributo di transazione utilizzando il Type Library editor. Per modificare l'attributo di transazione con il Type Library editor:

1 Scegliere il comando View | Type Library per attivare il Type Library editor.
2 Selezionare la classe corrispondente all'oggetto transazionale.
3 Fare clic sulla pagina COM+ e scegliere l'attributo di transazione desiderato.



Quando si imposta l'attributo di transazione, C++Builder inserisce uno speciale GUID per l'attributo specificato sotto forma di dato custom nella libreria di tipi. Questo valore non è riconosciuto al di fuori di C++Builder. Pertanto, ha un effetto solamente se si installa l'oggetto transazionale dall'IDE. In alternativa, questo valore può essere impostato da un amministratore di sistema utilizzando COM+ Component Manager o MTS Explorer.



Quando si modifica l'attributo di transazione, se l'oggetto transazionale è già installato, occorre prima disinstallarlo e successivamente reinstallarlo. Per eseguire queste operazioni, utilizzare i comandi Use Run | Install MTS objects o Run | Install COM+ objects.

Oggetti con e senza stato

Come qualsiasi altro oggetto COM, gli oggetti transazionali possono mantenere lo stato interno per diverse interazioni con un client. Per esempio, il client potrebbe impostare un valore di una proprietà durante una chiamata, e aspettarsi che quel valore di proprietà rimanga immutato alla chiamata successiva. Tale oggetto è detto essere **con stato**. Gli oggetti transazionali possono essere anche **senza stato**, il che vuol dire che non contengono stati intermedi durante l'attesa della chiamata successiva da parte di un client.

Quando una transazione viene completata o abbandonata, tutti gli oggetti che sono coinvolti nella transazione vengono disattivati, causando la perdita di qualsiasi stato acquisito nel corso della transazione. Questa operazione aiuta a garantire l'isolamento della transazione e la coerenza del database; inoltre, libera le risorse del server affinché siano utilizzate in altre transazioni. Il completamento di una transazione consente di reclamare le risorse utilizzate da un oggetto al momento della sua disattivazione. Per informazioni su come controllare quando viene rilasciato lo stato di una oggetto, consultare la sezione seguente.

Il mantenimento dello stato su un oggetto richiede che l'oggetto rimanga attivato, conservando risorse potenzialmente preziose come le connessione ad un database.

Come influenzare la modalità di completamento delle transazioni

Un oggetto transazionale utilizza i metodi di *IObjectContext* come mostrato nella tabella seguente per influenzare la modalità di completamento di una transazione.

Questi metodi, insieme all'attributo di transazione dell'oggetto, consentono di includere uno o più oggetti in un'unica transazione.

Tabella 44.1 Metodi IObjectContext per il supporto delle transazioni

Metodo	Descrizione
SetComplete	Indica che l'oggetto ha completato con successo le operazioni relative alla transazione. L'oggetto viene disattivato al ritorno dal metodo che prima ha immesso il contesto. L'oggetto viene riattivato alla successiva chiamata che richiede l'esecuzione dell'oggetto.
SetAbort	Indica che il lavoro dell'oggetto non potrà mai essere effettuato e la transazione dovrebbe essere ripristinata allo stato originale. L'oggetto viene disattivato al ritorno dal metodo che prima ha immesso il contesto. L'oggetto viene riattivato alla successiva chiamata che richiede l'esecuzione dell'oggetto.
EnableCommit	Indica che il lavoro dell'oggetto non è necessariamente eseguito, ma che gli aggiornamenti transazionali possono essere completati nella loro forma attuale. Utilizzare questo metodo per ritenere lo stato tra più chiamate successive di un client, consentendo al contempo il completamento delle transazioni. L'oggetto non viene disattivato finché non chiama SetComplete o SetAbort. EnableCommit è lo stato predefinito quando un oggetto viene attivato. Questo si verifica perché un oggetto *chiamerà sempre SetComplete o SetAbort prima di ritornare da un metodo*, a meno che l'oggetto non mantenga il suo stato interno per la chiamata successiva da un client.
DisableCommit	Indica che il lavoro dell'oggetto è inconsistente e che non può essere completato finché non vengono ricevute ulteriori chiamate del metodo dal client. Questo metodo deve essere chiamato prima di restituire il controllo al client per mantenere lo stato tra più chiamate successive del client mantenendo attiva al contempo la transazione corrente. DisableCommit impedisce la disattivazione dell'oggetto ed il rilascio delle sue risorse al ritorno da una chiamata a un metodo. Una volta che un oggetto ha chiamato DisableCommit, se un client cerca di effettuare la transazione prima che l'oggetto abbia chiamato EnableCommit o *SetComplete*, la transazione viene abbandonata.

Inizio di transazioni

Le transazioni possono essere controllate in tre modi:

- Possono essere controllate dal client.

  I client possono avere un controllo diretto sulle transazioni utilizzando un oggetto contesto di transazione (utilizzando l'interfaccia *ITransactionContext*).

- Possono essere controllate dal server.

  I server possono controllare le transazioni creando esplicitamente per esse un oggetto contesto. Quando il server crea un oggetto in tal modo, l'oggetto creato viene assimilato automaticamente nella transazione corrente.

- Le transazioni possono verificarsi automaticamente come risultato dell'attributo di transazione dell'oggetto.

Gli oggetti transazionali possono essere dichiarati in modo che i rispettivi oggetti siano sempre eseguiti all'interno di una transazione, indipendentemente da come gli oggetti sono stati creati. In questo modo, gli oggetti non devono includere alcuna logica per gestire le transazioni. Questa caratteristica riduce anche il carico sulle applicazioni client. I client non dovranno iniziare una transazione semplicemente perché il componente che stanno utilizzando lo richiede.

Impostazione di un oggetto transazionale sul lato client

Un'applicazione basata su client può controllare il contesto della transazione tramite l'interfaccia *ITransactionContextEx*. L'esempio di codice che segue mostra come un'applicazione client utilizza *CreateTransactionContextEx* per creare il contesto della transazione. Questo metodo restituisce un'interfaccia per questo oggetto.

```
#include <vcl\mtshlpr.h>

int main(int argc, char* argv[])
{
  // first, create a transactional object. [requires COM+ or local installation of MTS]

  TCOMITransactionClientExample first_client = CoTransactionClientExample::Create();

  //then, check to see if it's in a transaction.

  ITransactionContext* TCTX;
  HRESULT happily_transacting = first_client->QueryInterface(IID_ITransactionContext,
    (void**)&TCTX);

  // if it is in a transaction, you can do your data access calls from here
  // and then call Commit or Abort yourself instead of waiting for the
  // transactional object to do so.

  if (happily_transacting)
  {
    TVariant database = "name";
    TVariant record = "data";
    TVariant flag;
    first_client.UpdateData(&database, &record, &flag);
    flag ? TCTX->Commit() : TCTX->Abort();
  }
  else
  {
    // otherwise, you can create a transaction:
    ITransactionContextEx* TXCTX  = CreateTransactionContextEx();

    // and an object. any objects you create in this fashion
    // will be enlisted in the transaction represented by this object.

    TCOMITransactionClientExample* second_client;
    TXCTX->CreateInstance(CLSID_TransactionClientExample, IID_ITransactionClientExample,
      (void**)&second_client);

    // and then perform your data access and commit or abort.

    TVariant database = "name";
    TVariant record = "data";
    TVariant flag;
    second_client->UpdateData(&database, &record, &flag);
    flag ? TXCTX->Commit() : TXCTX->Abort();
```

```
    }
    return 0;
  }
```

Impostazione di un oggetto transazionale sul lato server

Per controllare il contesto della transazione sul lato server, si crea un'istanza di *ObjectContext*. Nell'esempio seguente il metodo Transfer si trova nell'oggetto transazionale. Usando *ObjectContext* in questo modo, l'istanza creata dell'oggetto erediterà tutti gli attributi di transazione dell'oggetto che la crea.

```
#include <vcl\mtshlpr.h>

STDMETHODIMP TTransactionServerExampleImpl::DoTransactionContext(long execflag)
{
  if (m_spObjectContext->IsInTransaction())
  {

    // this means the current object has a transaction, and can pass
    // its transaction information to its children.
    // for simplicity, this object simply creates another object of its
    // own type, within the same transaction.
    // NOTE: you are still responsible for aggregating, if appropriate;

    if (execflag)
    {
      TCOMITransactionServerExample* inner;
      m_spObjectContext->CreateInstance(CLSID_TransactionServerExample,
                                  IID_ITransactionServerExample, (void**)&inner);
      inner->DoTransactionContext(false);

      // add data access code here. data_access_succeeded() below is
      // an unimplemented placeholder.

      data_access_succeeded() ? m_spObjectContext->EnableCommit()
                              : m_spObjectContext->DisableCommit();
    }
  }
  else
  {
    //this means the current object has no transaction, and must
    //create one the way a client would.

    ITransactionContextEx* TCTX = CreateTransactionContextEx();
    TCOMITransactionServerExample* inner;

    // afterwards, follow the same steps.

    TCTX->CreateInstance(CLSID_TransactionServerExample,
                         IID_ITransactionServerExample, (void**)&inner);
    inner->DoTransactionContext(true);

    // add data access code here. data_access_succeeded() below is
    // an unimplemented placeholder.
    data_access_succeeded() ? TCTX->Commit() : TCTX->Abort();
  }
}
```

Durata della transazione

La durata della transazione imposta per quanto tempo (in secondi) può rimanere attiva una transazione. Il sistema annulla automaticamente le transazioni che risultano ancora attive alla scadenza di tale intervallo temporale. Per impostazione predefinita, il valore della durata è 60 secondi. È possibile disattivare la durata delle transazioni specificando un valore 0, il che è utile durante il debug degli oggetti transazionali.

Per impostare il valore di time-out sul computer:

1 In MTS Explorer o in COM+ Component Manager, selezionare Computer, My Computer.

   Per impostazione predefinita, My Computer corrisponde al computer locale.

2 Fare clic con il tasto destro e scegliere Properties e quindi la pagina Options.

   La pagina Options viene usata per impostare la proprietà di time-out della transazione del computer.

3 Cambiare il valore di time-out in 0 per disattivare la durata delle transazioni.

4 Fare clic su OK per salvare l'impostazione.

Per ulteriori informazioni sul debug di applicazioni MTS, consultare "Debug e collaudo di oggetti transazionali" a pagina 44-28.

Sicurezza basata sul ruolo

Attualmente MTS e COM+ forniscono la sicurezza basata sul ruolo, che prevede appunto l'assegnazione di un ruolo ad un gruppo logico di utenti. Per esempio, un'applicazione di informazioni mediche potrebbe definire i ruoli dei medici, dei radiologi e dei pazienti.

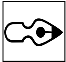
Si deve definire l'autorizzazione per ciascun oggetto e interfaccia assegnando i ruoli. Per esempio, nell'applicazione per i medici, solo il medico può essere autorizzato a vedere tutti i record clinici, mentre il radiologo può vedere solo le radiografie e i pazienti possono vedere solo i propri record.

Normalmente, i ruoli vengono definiti durante lo sviluppo dell'applicazione e vengono assegnati a ciascun package MTS o applicazione COM+. Questi ruoli vengono quindi assegnati agli specifici utenti quando l'applicazione viene distribuita. Gli amministratori possono configurare i ruoli usando MTS Explorer o COM+ Component Manager.

Se si vuole controllare l'accesso per bloccare il codice invece dell'intero oggetto, è possibile introdurre una sicurezza ancora più mirata utilizzando il metodo *IsCallerInRole* di *IObjectContext*. Tale metodo funziona sole se la sicurezza è attivata, cosa che può essere controllata chiamando il metodo *IsSecurityEnabled* di *IObjectContext*. Ad esempio:

```
if (m_spObjectContext.IsSecurityEnabled()) // check if security is enabled
```

```
{
  if (!m_spObjectContext.IsCallerInRole("Physician")) // check caller's role
  { // If not a physician, do something appropriate here.
  }
  else
  { // execute the call normally
  }
}
else // no security enabled
{ // do something appropriate
}
```



Per le applicazioni che necessitano di una sicurezza più forte, gli oggetti contesto implementano l'interfaccia *ISecurityProperty*, i cui metodi consentono il reperimento dell'identificatore di sicurezza di Window (SID) del chiamante diretto e del creatore dell'oggetto, oltre che del SID dei client che stanno utilizzando l'oggetto.

Panoramica sulla creazione di oggetti transazionali

Il processo di creazione di un oggetto transazionale è il seguente:

1 1 Usare Transactional Object wizard per creare un oggetto transazionale.
2 Aggiungere i metodi e le proprietà all'interfaccia dell'oggetto usando il Type Library editor. Per informazioni dettagliate sull'aggiunta di metodi e proprietà usando il Type Library editor, consultare il Capitolo 39, "Funzionamento delle librerie di tipi".
3 Durante l'implementazione dei metodi dell'oggetto è possibile utilizzare l'interfaccia *IObjectContext* per gestire le transazioni, lo stato permanente e la sicurezza.Inoltre, se si passano riferimenti a oggetti, è necessario porre particolare attenzione affinché siano gestiti in modo corretto. (Consultare "Passaggio di riferimenti a oggetti" on page 27.)
4 Correggere e collaudare l'oggetto transazionale.
5 Installare l'oggetto transazionale in un package MTS o in un'applicazione COM+.
6 Amministrare gli oggetti usando MTS Explorer o COM+ Component Manager.

Uso di Transactional Object wizard

Transactional Object wizard viene utilizzato per creare oggetti COM in grado di sfruttare i vantaggi derivanti dalla gestione delle risorse, dall'elaborazione delle transazioni e dalla sicurezza basata sui ruoli, messi a disposizione da MTS o COM+.

Per visualizzare Transactional Object wizard:

1 Scegliere il comando File|New|Other.
2 Selezionare la pagina ActiveX.
3 Fare doppio clic sull'icona Transactional Object.

Nel wizard si deve specificare quanto segue:

- Un modello di thread che indichi come le applicazioni client possano chiamare l'interfaccia dell'oggetto. Il modello di thread scelto determina come viene registrato l'oggetto. Chi programma deve accertarsi che l'implementazione dell'oggetto sia coerente con il modello selezionato. Per ulteriori informazioni sui modelli di thread, consultare "Scelta di un modello di threading per un oggetto transazionale" a pagina 44-19.
- Un modello transazionale
- Una indicazione del fatto che l'oggetto notifichi o meno gli eventi ai client. Il supporto degli eventi è previsto solo per eventi tradizionali, non per eventi COM.

Una volta completata questa procedura, viene aggiunta una nuova unit al progetto attivo che contiene la definizione per l'oggetto transazionale. Inoltre, il wizard aggiunge un progetto di libreria di tipi e la apre nel Type Library editor. A questo punto è possibile mostrare le proprietà e i metodi dell'interfaccia tramite la libreria di tipi. L'interfaccia viene definita come se si stesse definendo un qualsiasi oggetto COM, secondo le modalità descritte in "Definizione dell'interfaccia di un oggetto COM" a pagina 41-9.

L'oggetto transazionale implementa un'**interfaccia duale**, che supporta sia il binding anticipato (in fase di compilazione) tramite vtable, sia il binding differito (in fase di esecuzione) tramite l'interfaccia *IDispatch*.

L'oggetto transazionale generato implementa i metodi dell'interfaccia *IObjectControl*, *Activate*, *Deactivate* e *CanBePooled*.

Non è strettamente necessario utilizzare il Transactional Object wizard. È possibile convertire qualsiasi oggetto Automation in un oggetto transazionale COM+ (e qualsiasi oggetto Automation in-process in un oggetto transazionale MTS) utilizzando la pagina COM+ del Type Library editor ed installando poi l'oggetto in un package MTS o in un'applicazione COM+. Tuttavia, il Transactional Object wizard offre alcuni benefici:

- Implementa automaticamente l'interfaccia *IObjectControl*, aggiungendo all'oggetto gli eventi *OnActivate* e *OnDeactivate*, in modo da consentire la creazione di gestori di evento che rispondano al momento dell'attivazione o disattivazione dell'oggetto.
- Genera automaticamente un membro m_spObjectContext semplificando così all'oggetto l'accesso ai metodi di *IObjectContext* per controllare l'attivazione e le transazioni.

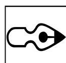
Scelta di un modello di threading per un oggetto transazionale

L'ambiente di runtime MTS o COM+ gestisce automaticamente i thread. Gli oggetti transazionali non creano thread. Inoltre non devono mai interrompere un thread che effettua una chiamata in una DLL.

Quando nel Transactional object wizard si specifica il modello di threading, si specifica come gli oggetti vengono assegnati ai thread per l'esecuzione dei metodi.

Tabella 44.2 Modelli di threading per oggetti transazionali

Modello di threading	Descrizione	Vantaggi e svantaggi dell'implementazione
Single	Non supporta alcun thread. Le richieste del client vengono serializzate tramite il meccanismo di chiamata. Tutti gli oggetti di un componente single-threaded vengono eseguiti sul thread principale. Questo è compatibile con il modello di threading COM predefinito, che viene usato per i componenti che non hanno un attributo Threading Model Registry o per i componenti COM che non sono rientranti. L'esecuzione del metodo viene serializzata in tutti gli oggetti del componente e in tutti i componenti di un processo.	Consente ai componenti di usare le librerie che non sono rientranti. Scalabilità molto limitata. I componenti a thread singolo con stato sono soggetti a blocchi critici. È possibile eliminare questo problema usando oggetti senza stato e chiamando *SetComplete* prima di ritornare da un qualsiasi metodo.
Apartment (or Single-threaded apartment)	Ogni oggetto viene assegnato ad un thread, che vale per l'intera durata dell'oggetto; comunque, si possono usare più thread per più oggetti. Questo è un modello COM standard di simultaneità. Ogni apartment viene collegato ad un specifico thread ed ha una sorgente di messaggi Windows.	Fornisce miglioramenti significativi rispetto alla concorrenza nel modello di threading Single. Due oggetti possono essere eseguiti contemporaneamente finché non svolgono la stessa attività. Simile ad un apartment COM, tranne per il fatto che gli oggetti possono essere distribuiti su più processi.
Both	Identico ad Apartment, tranne per il fatto che le calback ai client sono serializzate.	Stessi vantagggi di Apartment. Inoltre, questo modello è necessario nel caso si voglia usare l'Object Pooling.



Questi modelli di threading sono simili a quelli definiti dagli oggetti COM. Tuttavia, poiché l'ambiente MTS e COM+ forniscono un maggiore supporto di base per i thread, il significato di ciascun modello di threading qui è diverso. Inoltre, i modelli free threading non si applicano agli oggetti transazionali a causa del supporto residente delle attività.

Attività

Oltre al modello di threading, gli oggetti transazionali supportano la concorrenzialità tramite le **attività**. Le attività sono registrate nel contesto dell'oggetto e l'associazione

tra un oggetto ed un'attività non può essere cambiata. Un'attività include l'oggetto transazionale creato dal client base, oltre a tutti gli oggetti MTS creati da quell'oggetto e dai suoi discendenti. Questi oggetti possono essere distribuiti in uno o più processi, che vengono eseguiti su uno o più computer.

Per esempio, l'applicazione clinica per i medici può avere un oggetto transazionale per aggiungere gli aggiornamenti e rimuovere i record ai vari database medici, ciascuno rappresentato da un oggetto diverso. Questo record aggiunto può usare anche altri oggetti, come un oggetto ricezione per registrare la transazione. Ciò genera diversi oggetti transazionali che sono direttamente o indirettamente sotto il controllo di un client base. Questi oggetti appartengono tutti alla stessa attività.

MTS o COM+ tengono la traccia del flusso d'esecuzione in ciascuna attività, impedendo così che il parallelismo involontario danneggi lo stato dell'applicazione. Questa caratteristica genera un singolo thread logico di esecuzione per tutta una serie di oggetti potenzialmente distribuiti. Avendo un thread logico, le applicazioni sono notevolmente più facili da scrivere.

Quando un oggetto transazionale viene creato da un contesto esistente, tramite un oggetto contestuale di transazione o un contesto di oggetto, il nuovo oggetto diventa un membro della stessa attività. In altre parole, il nuovo contesta eredita l'identificatore di attività del contesto usato per crearlo.

È permesso solo un singolo thread logico di esecuzione all'interno dell'attività. Questo comportamento è simile a quello di un modello di threading COM di tipo apartment, tranne per il fatto che gli oggetti possono essere distribuiti in più processi. Quando un client base chiama in un'attività, tutte le altre richieste di lavoro dell'attività (provenienti, per esempio, da un altro thread del client) vengono bloccate finché il thread iniziale di esecuzione non ritorna di nuovo al client.

In ambiente MTS, ogni oggetto transazionale appartiene a un'attività. In ambiente COM+, è possibile configurare il modo in cui l'oggetto partecipa alle attività impostando la **sincronizzazione della chiamata**. Sono disponibili le opzioni seguenti:

Tabella 44.3 Opzioni di sincronizzazione della chiamata

Opzione	Significato
Disabled	COM+ non assegna attività all'oggetto ma può ereditarle col contesto del chiamante. Se il chiamante non ha alcuna transazione o contesto di oggetto, l'oggetto non viene assegnato a un'attività. Il risultato è come se l'oggetto non fosse installato nell'applicazione COM+. Questa opzione non si dovrebbe utilizzare se un qualsiasi oggetto nell'applicazione utilizza un manager di risorse o se l'oggetto supporta transazioni o attivazione just-in-time.
Not Supported	COM+ non assegna mai l'oggetto a un'attività, indipendentemente dallo stato del suo chiamante. Questa opzione non si dovrebbe utilizzare se un qualsiasi oggetto nell'applicazione utilizza un manager di risorse o se l'oggetto supporta transazioni o l'attivazione just-in-time.
Supported	COM+ assegna l'oggetto alla stessa attività del suo chiamante. Se il chiamante non appartiene a un'attività, anche l'oggetto si comporterà in modo analogo. Questa opzione non si dovrebbe utilizzare se un qualsiasi oggetto nell'applicazione utilizza un manager di risorse o se l'oggetto supporta transazioni o l'attivazione just-in-time.

Tabella 44.3 Opzioni di sincronizzazione della chiamata

Opzione	Significato
Required	COM+ assegna sempre l'oggetto a un'attività, creandone una se occorre. Questa opzione si deve utilizzare se l'attributo di transazione è Supported o Required.
Requires New	COM+ assegna sempre l'oggetto a una nuova attività, distinta da quella del suo chiamante.

Generazione di eventi in ambiente COM+

Molte tecnologie basate su COM, come il motore di script di ActiveX e i controlli ActiveX, utilizzano i convogliatori di eventi e le interfacce dei punti di connessione di COM per generare eventi. I convogliatori di eventi e i punti di connessione sono esempi di un modello ad eventi molto unito. In tale modello, le applicazioni che generano eventi (detti publisher nella terminologia COM+ e convogliatori nella precedente terminologia COM) sono consapevoli di quali sono le applicazioni che rispondono agli eventi (dette subscriber), e vice versa. La durata dei publisher e dei subscriber coincide; essi devono essere attivi contemporaneamente. La raccolta di subscriber e il meccanismo che notifica loro il verificarsi degli eventi, devono essere gestiti e implementati nel publisher.

COM+ ha introdotto un nuovo sistema per la gestione degli eventi. Invece di appesantire ogni publisher con la gestione e con la notifica di ciascun subscriber, interviene il sistema sottostante (COM+) e prende il controllo di questo processo. Il modello a eventi di COM+ è scarsamente interdipendente, consentendo lo sviluppo, la distribuizione e l'attivazione di publisher e subscriber in modo indipendente l'uno dall'altro.

Benché il modello a eventi di COM+ semplifichi notevolmente la comunicazione fra publisher e subscriber, introduce anche alcune attività amministrative aggiuntive per gestire il nuovo strato di software che ora esiste tra di loro. Le informazioni sugli eventi e sui subscriber sono conservate in una parte del COM+ Catalog detta anche event store. Per eseguire questi compiti amministrativi vengono utilizzati strumenti come Component Services Manager. Lo strumento Component Services supporta completamente gli script, consentendo l'automazione di gran parte dei compiti amministrativi. Ad esempio, una procedura di installazione potrebbe eseguire queste operazioni durante la sua esecuzione. Inoltre, l'event store può essere gestito da programma utilizzando l'oggetto *TComAdminCatalog*. I componenti COM+ possono essere installati anche direttamente da C++Builder, selezionando il comando Run | Install COM+ object.

Come nel caso del modello a eventi strettamente interdipendenti, un evento è semplicemente un metodo in un'interfaccia. Pertanto, è necessario per prima cosa creare un'interfaccia per i metodi dell'evento. È possibile utilizzare COM+ Event Object wizard di C++ Builder per creare un progetto contenente un oggetto evento di COM+. Quindi, utilizzando lo strumento amministrativo Component Services (o *TComAdminCatalog* o l'IDE), creare un'applicazione COM+ che contenga un componente della classe evento. Quando si crea nell'applicazione COM+ il

componente della classe evento, si selezionerà l'oggetto evento. La classe evento è il collante utilizzato da COM+ per collegare il publisher alla lista dei subscriber.

La cosa interessante su un oggetto evento di COM+ è che esso non contiene alcuna implementazione dell'interfaccia dell'evento. Un oggetto evento di COM+ definisce semplicemente l'interfaccia che utilizzeranno i publisher e i subscriber per comunicare. Quando si crea con C++Builder un oggetto evento di COM+, per definire l'interfaccia si utilizzerà il Type Library editor. L'interfaccia viene implementata quando si crea un'applicazione COM+ e il relativo componente della classe evento. La classe evento, quindi, contiene un riferimento e consente laccesso all'implementazione fornita da COM+. In esecuzione, il publisher crea un'istanza della classe evento con i soliti meccanismi COM (per esempio *CoCreateInstance*). L'implementazione COM+ dell'interfaccia è tale per cui tutto ciò che un publisher deve fare è chiamare un metodo dell'interfaccia (attraverso l'istanza della classe evento) per generare un evento che notoficherà a tutti i subscriber.

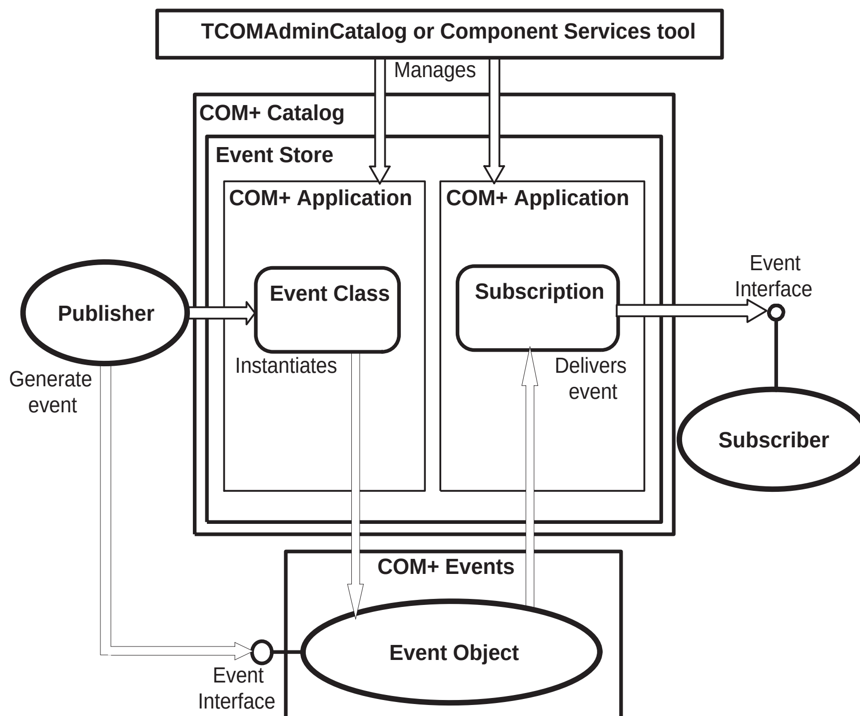
Nota: Un publisher non deve essere necessariamente esso stesso un componente COM. Un publisher è semplicemente una qualsiasi applicazione che crea un'istanza della classe evento e genera eventi chiamando i metodi presenti nell'interfaccia dell'evento.

Anche il componente subscriber deve essere installato nel COM+ Catalog. Anche in questo caso, l'operazione può essere svolta sia da programma utilizzando *TComAdminCatalog*, sia dall'IDE sia mediante lo strumento amministrativo gestione Component Services. Il componente subscriber può essere installato in un'applicazione separata COM+ oppure può essere installato nella stessa applicazione utilizzata per contenere il componente della classe evento. Dopo avere installato il componente, per ogni interfaccia di evento supportata dal componente si deve creare una subscription. Dopo avere creato la subscription, selezionare quelle classi evento (cioè i publisher) che si desidera siano ricevute dal componente. Un componente subscriber può selezionare singole classi evento oppure tutte le classi evento.

A differenza dell'oggetto evento di COM+, un oggetto subscription di COM+ contiene la propria implementazione di un'interfaccia di evento; questa è la posizione in cui viene svolto il lavoro effettivo per rispondere all'evento nel momentoin cui viene generato. Il COM+ Event Subscription Object wizard di C++ Builder può essere utilizzato per creare un progetto che contiene un componente subscriber.

La figura seguente rappresenta l'interazione fra publisher, subscriber e il Catalog di COM+:

Figura 44.1 Il sistema a eventi di COM+



Uso di Event Object wizard

È possibile creare oggetti event utilizzando Event Object wizard. Il wizard per prima cosa controlla se il progetto corrente contiene del codice di implementazione, in quanto i progetti che contengono oggetti evento di COM+ non includono un'implementazione. Possono contenere solamente definizioni dell'oggetto evento.(In un singolo progetto è possibile, tuttavia, includere più oggetti evento di COM+)

Per attivare Event Object wizard:

1. Scegliere il comando File|New |Other.
2. Selezionare la pagina ActiveX.
3. Fare doppio clic sull'icona COM+ Event Object.

Nell'Event Object wizard, specificare il nome dell'oggetto evento, il nome dell'interfaccia che definisce i gestori di evento e (facoltativamente) una breve descrizione degli eventi.

All'uscita, il wizard crea un progetto contenente una libreria di tipi che definisce l'oggetto evento e la sua interfaccia. Utilizzare il Type Library editor per definire i metodi di quell'interfaccia. Questi metodi sono i gestori di evento che i client implementano per rispondere agli eventi.

Il progetto dell'oggetto evento include il file di progetto, la unit _ATL per importare le classi del modello ATL, e la unit _TLB per definire le informazioni della libreria di tipi. Non include una unit di implementazione, comunque, in quanto gli oggetti evento di COM+ non hanno implementazione. L'implementazione dell'interfaccia è a carico del client. Quando l'oggetto server chiama un oggetto evento di COM+, COM+ intercetta la chiamata e l'invia ai client registrati.Poiché gli oggetti evento di COM+ non richiedono l'oggetto implementazione, tutto ciò che si deve fare dopo avere definito l'interfaccia dell'oggetto nel Type Library editor è compilare il progetto e installarlo in COM+.

COM+ impone alcune restrizioni sulle interfacce degli oggetti evento. L'interfaccia che si definisce nel Type Library editor per l'oggetto evento deve rispettare le seguenti regole:

- L'interfaccia dell'oggetto evento deve derivare da IDispatch.
- Tutti i nomi di metodo devono essere unici in tutte le interfacce dell'oggetto evento.
- Tutti i metodi sull'interfaccia dell'oggetto evento devono restituire un valore HRESULT.
- Il modificatore per tutti i parametri dei metodi deve essere [in].

Uso di COM+ Event Subscription object wizard

È possibile creare il componente subscriber utilizzando C++Builder COM+ Subscription Object wizard. Per attivare il wizard:

1. Scegliere il comando File|New | Other.
2. Selezionare la pagina ActiveX.
3. Fare doppio clic sull'icona COM+ Subscription Object.

Nella finestra di dialogo del wizard, immettere il nome della classe che implementerà l'interfaccia dell'evento. Scegliere dalla casella combinata il modello di threading. Nel campo Interface, è possibile immettere il nome dell'interfaccia dell'evento, oppure è possibile fare clic sul pulsante Browse per attivare una lista di tutte le classi evento attualmente installate nel Catalog di COM+. Anche la finestra di dialogo COM+ Event Interface Selection contiene un pulsante Browse. È possibile utilizzare questo pulsante per cercare e selezionare una libreria di tipi che contenga l'interfaccia dell'evento. Quando si seleziona una classe evento esistente (o si immette la libreria), il wizard darà la possibilità di implementare automaticamente l'interfaccia supportata da quella classe evento. Se si seleziona la casella di controllo Implement Existing Interface, il wizard creerà automaticamente tutti i metodi nell'interfaccia. È possibile decidere di fare implementare al wizard le interfacce ereditate selezionando la casella di controllo Implement Ancestor Interfaces. Tre interfacce dell'antenato non

vengono mai implementate dal wizard: *IUnknown*, *IDispatch* e *IAppServer*. Per completare il wizard, immettere una breve descrizione del componente subscriber dell'evento e fare clic su OK.

Attivazione di eventi utilizzando un oggetto evento di COM+

Per generare un evento, un publisher crea per prima cosa un'istanza della classe evento, con i soliti meccanismi COM (per esempio *CoCreateInstance*). È bene ricordare che la classe evento contiene la propria implementazione dell'interfaccia dell'evento, e pertanto la generazione di un evento non è altro che la semplice chiamata del metodo appropriato dell'interfaccia.

A questo punto subentra il sistema a eventi di COM+. La chiamata al metodo di un evento fa sì che il sistema cerchi tutti i subscriber nel Catalog di COM+ e invii loro notifica dell'evento. Dal lato del subscriber, l'evento non sembra essere niente di più di una chiamata al metodo dell'evento.

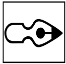
Quando un publisher genera un evento, i subscriber ne vengono informati in modo sincrono, uno alla volta. Non v'è alcun modo per specificare l'ordine di notifica, né si può contare sul fatto che l'ordine sia lo stesso ogni volta che viene generato un evento. Quando una classe evento è installata nel Catalog di COM+, l'amministratore può selezionare l'opzione *FireInParallel* per richiedere che l'evento venga notificato utilizzando più thread. Ciò non garantisce la consegna simultanea; è semplicemente una richiesta al sistema di consentire che ciò possa avvenire.

Il valore restituito al publisher è un'aggregazione di tutti i codici restituiti da ogni subscriber. Il publisher non ha alcun modo diretto per scoprire quale subscriber non è andato a buon fine. Per fare ciò, un publisher deve implementare un filtro di publisher. Per ulteriori informazioni su questo argomento, consultare la documentazione MSDN di Microsoft. La tabella seguente riassume i codici derivanti dalla scelta di varie combinazioni:

Tabella 44.4 Codici di ritorno dei publisher di evento

Codice restituito	Significato
S_OK	Tutti i subscriber sono andati a buon fine.
EVENT_S_SOME_SUBSCRIBERS_FAILED	Non è stato possibile chiamare qualche subscriber, oppure alcuni hanno restituito un codice di malfunzionamento (notare che questa non è una condizione di errore).
EVENT_E_ALL_SUBSCRIBERS_FAILED	Non è stato possibile chiamare alcun subscriber oppure tutti i subscriber hanno restituito un codice di malfunzionamento.
EVENT_S_NOSUBSCRIBERS	Nel Catalog di COM+ non c'è alcuna subscription (notare che questa non è una condizione di errore).

Passaggio di riferimenti a oggetti



Le informazioni sul passaggio di riferimenti a oggetti sono valide solo per MTS, non per COM+. Questo meccanismo è necessario in ambiente MTS perché è necessario garantire che tutti i puntatori agli oggetti che sono in esecuzione in ambiente MTS vengano indirizzati attraverso gli interceptor. Poiché gli interceptor sono incorporati in COM+, non è necessario passare i riferimenti a oggetti.

In ambiente MTS, è possibile passare riferimenti a oggetti (da usare, per esempio, come callback), solo nei seguenti modi:

- Tramite il ritorno da un'interfaccia di creazione dell'oggetto, come, per esempio, *CoCreateInstance* (o un suo equivalente), *ITransactionContext::CreateInstance* o *IObjectContext::CreateInstance.*
- Tramite una chiamata a *QueryInterface*.
- Tramite un metodo che ha chiamato *SafeRef* per ottenere il riferimento oggetto.

Un riferimento a oggetto ottenuto in uno dei modi sopra riportati viene definito **riferimento sicuro**. I metodi chiamati usando riferimenti sicuri saranno eseguiti con assoluta sicurezza all'interno del contesto corretto.

L'ambiente di esecuzione di MTS vuole che le chiamate utilizzino riferimenti sicuri, in modo da gestire le opzioni di contesto e consentire agli oggetti transazionali di avere una durata indipendente dai riferimenti del client. I riferimenti sicuri non sono necessari in ambiente COM+.

Uso del metodo SafeRef

Un oggetto può usare una funzione *SafeRef* in modo da ottenere un riferimento sicuro a se stesso che consenta di oltrepassare il contesto. Questa funzione è disponibile come metodo dell'oggetto *TMtsDll*; essa controlla se il server è in esecuzione in ambiente MTS o COM+ e restituisce, in base a ciò, il puntatore opportuno.

SafeRef accetta come input

- Un riferimento all'ID (RIID) dell'interfaccia che l'attuale oggetto vuole passare ad un altro oggetto o client.
- AUn riferimento all'interfaccia IUnknown dell'oggetto corrente.

SafeRef restituisce un puntatore all'interfaccia specificata nel parametro RIID che rende certo il passaggio al di là del contesto dell'oggetto attivo. Questo puntatore restituisce NULL se l'oggetto richiede un riferimento sicuro su un oggetto diverso da se stesso, o se l'interfaccia richiesta nel parametro RIID non viene implementata.

Quando un oggetto MTS vuole passare un riferimento a se stesso per un client o per un altro oggetto (da usare, per esempio, come callback), prima chiamerà sempre *SafeRef* e dopo passerà il riferimento restituito da questa chiamata. Un oggetto non passerà mai un puntatore **self**, o un riferimento a se stesso ottenuto tramite una chiamata interna a *QueryInterface*, ad un client o a qualsiasi altro oggetto. Una volta che un tale riferimento è stato passato fuori dal contesto dell'oggetto, non è più un riferimento valido.

La chiamata a *SafeRef* su un riferimento che è già sicuro restituisce il riferimento stesso non modificato, tranne che il numero di riferimento sull'interfaccia non viene incrementato.

Quando un client chiama QueryInterface su un riferimento sicuro, il riferimento restituito al client è un riferimento sicuro.

Un oggetto che ottiene un riferimento sicuro deve rilasciare tale riferimento quando non gli occorre più.

Per informazioni dettagliate su *SafeRef*, consultare l'argomento SafeRef nella documentazione Microsoft.

Callback

Gli oggetti possono effettuare il callback ai client e agli altri oggetti transazionali. Per esempio, è possibile avere un oggetto che crei un altro oggetto. La creazione di un oggetto può passare un riferimento di se stesso all'oggetto creato; tale oggetto può usare poi questo riferimento per chiamare l'oggetto creante.

Se si sceglie di usare i callback, si devono tenere presenti le seguenti restrizioni:

- Una chiamata di ritorno al client base o ad un altro package richiede la sicurezza a livello di accesso sul client. Inoltre, il client deve essere un server DCOM.
- I firewall intermedi potrebbero bloccare le chiamate di ritorno al client.
- Il lavoro fatto sul callback viene eseguito nell'ambiente dell'oggetto da chiamare. Può far parte della stessa transazione, di una transazione diversa o di nessuna transazione.
- In ambiente MTS, l'oggetto che crea deve chiamare *SafeRef* e passare il riferimento restituito all'oggetto creato in modo da effettuare una chiamata di ritorno a se stesso.

Debug e collaudo di oggetti transazionali

È possibile effettuare il debug degli oggetti transazionali locali e remoti. Durante il debug di oggetti transazionali, può essere opportuno disattivare il timeout delle transazioni.

Il timeout della transazione imposta per quanto tempo (in secondi) può rimanere attiva una transazione. Le transazioni che sono ancora attive dopo il timeout vengono abbandonate automaticamente dal sistema. Per impostazione predefinita, il valore del timeout è di 60 secondi. È possibile disattivare il timeout delle transazioni specificando un valore 0, il che è utile durante il debug.

Per informazioni sul debug remoto, consultare l'argomento Remote Debugging nella Guida in linea.

Quando si effettua il collaudo di un oggetto transazionale che si intente eseguire in ambiente MTS, può essere opportuno verificare prima l'oggetto asternamente all'ambiente MTS per semplificare l'ambiente di verifica.

Durante lo sviluppo di un server, non si può effettuare di nuovo il build del server mentre è ancora in memoria, in quanto si riceverebbe un errore del compilatore del tipo "Cannot write to DLL while executable is loaded". Per evitare ciò, si possono impostare le proprietà del package MTS o dell'applicazione COM+ in modo da disattivare il server quando è in stato di attesa.

Per disattivare il server quando è in stato di attesa:

1 In MTS Explorer o in COM+ Component Manager, fare clic destro sul package MTS o sull'applicazione COM+ in cui è installato l'oggetto transazionale e scegliere il comando Properties.
2 Selezionare la pagina Advanced.

   Advanced determina se il processo del server associato ad un package viene eseguito sempre o se viene disattivato dopo un certo lasso di tempo.
3 Cambiare il valore di timeout su 0, il che disattiva il server quando non ha più un client a cui fornire un servizio.
4 Fare clic su OK per salvare l'impostazione.

Installazione di oggetti transazionali

Le applicazioni MTS sono costituite da un gruppo di oggetti MTS in-process che vengono eseguiti in un'unica istanza dell'eseguibile MTS (EXE). Un gruppo di oggetti COM che vengono eseguiti tutti nello stesso processo viene chiamato **package**. Una singola macchina può eseguire molti package differenti, dove ciascun package è in esecuzione all'interno di un EXE MTS distinto.

In ambiente COM+, si opera con un gruppo analogo detto applicazione COM+. In un'**applicazione COM+** gli oggetti non devono essere necessariamente interni al processo e non esiste un ambiente di esecuzione separato.

È possibile raggruppare i componenti dell'applicazione in un singolo package MTS o in un'applicazione COM+ da gestire all'interno di un unico processo. Può essere opportuno distribuire i componenti in più package MTS o applicazioni COM+ in modo da suddividere l'applicazione in più processi o macchine.

Per installare oggetti transazionali in un package o in un'applicazione COM+:

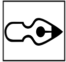
1 Se il sistema supporta COM+, scegliere l'opzione Run|Install COM+ objects. Se il sistema non supporta COM+ ma sul sistema è installato MTS, scegliere l'opzione Run|Install MTS objects. Se il sistema supporta né MTS né COM+, non si vedrà alcuna opzione di menu per installare gli oggetti transazionali.
2 Nella finestra di dialogo Install Object, selezionare gli oggetti da installare.
3 Se si stanno installando oggetti MTS, fare clic sul pulsante Package per ottenere un elenco dei package MTS nel sistema. Se si stanno installando oggetti COM+, fare clic sul pulsante Application. Indicare il package MTS o l'applicazione COM+ in cui si stanno installando gli oggetti. È possibile scegliere l'opzione Into New Package o Into New Application per creare un nuovo package MTS o un'applicazione COM+ in cui installare l'oggetto. È possibile anche scegliere Into

Existing Package o Into Existing Application per installare l'oggetto in un package MTS o in un'applicazione COM+ esistente fra quelli elencati.

4 Scegliere OK per aggiornare il catalogo, rendendo così gli oggetti disponibili in fase di esecuzione.

I package MTS possono contenere componenti di più DLL, e i componenti di una singola DLL possono essere installati in package differenti. Tuttavia, un singolo componente non può essere distribuito in più package.

Analogamente, le applicazioni COM+ possono contenere componenti inclusi in più eseguibili, e componenti differenti di un singolo eseguibile possono essere installati in applicazioni COM+ differenti.



È possibile anche installare l'oggetto transazionale utilizzando COM+ Component Manager o MTS Explorer. Durante l'installazione dell'oggetto con uno di questi strumenti, accertarsi di applicare le impostazioni per l'oggetto che appaiono sulla pagina COM+ del Type Library editor. Queste impostazioni non vengono applicate automaticamente quando non si effettua l'installazione dall'IDE.

Amministrazione di oggetti transazionali

Una volta installati degli oggetti transazionali, è possibile amministrare questi oggetti di runtime usando MTS Explorer (se sono installati in un package MTS) o COM+ Component Manager (se sono installati in un'applicazione COM+). I due strumenti sono identici, tranne per il fatto che MTS Explorer opera nell'ambiente di esecuzione di MTS e COM+ Component Manager opera su oggetti COM+.

COM+ Component Manager e MTS Explorer posseggono un'interfaccia utente grafica per la gestione e la distribuzione di oggetti transazionali. Utilizzando uno di questi strumenti è possibile

- Configurare oggetti transazionali, package MTS o applicazioni COM+, e ruoli
- Visualizzare le proprietà dei componenti in un package o in un'applicazione COM+ e visualizzare i package MTS o le applicazioni COM+ installati su un computer
- Controllare e gestire le transazioni per gli oggetti che comprendono transazioni
- Spostare i package MTS o le applicazioni COM+ da un computer all'altro
- Rendere un oggetto transazionale remoto disponibile per un client locale

Per informazioni dettagliate su questi strumenti, consultare l'appropriato manuale *Administrator's Guide* di Microsoft.

Parte

V

Creazione di componenti custom

I capitoli inclusi nella sezione "Creazione di componenti custom" presentano i concetti e le tecniche necessarie per progettare e implementare componenti custom in C++Builder.

Capitolo

45

Introduzione alla creazione di componenti

Questo capitolo offre una panoramica sulla progettazione di componenti e sulla procedura per la scrittura di componenti per applicazioni C++Builder. Presuppone che si abbia familiarità con C++Builder e con i suoi componenti standard.

- Librerie di classi
- Componenti e classi
- Creazione di componenti
- Cosa mettere in un componente
- Creazione di un nuovo componente
- Verifica di componenti non installati
- Collaudo di componenti installati
- Installazione di un componente sulla Component palette

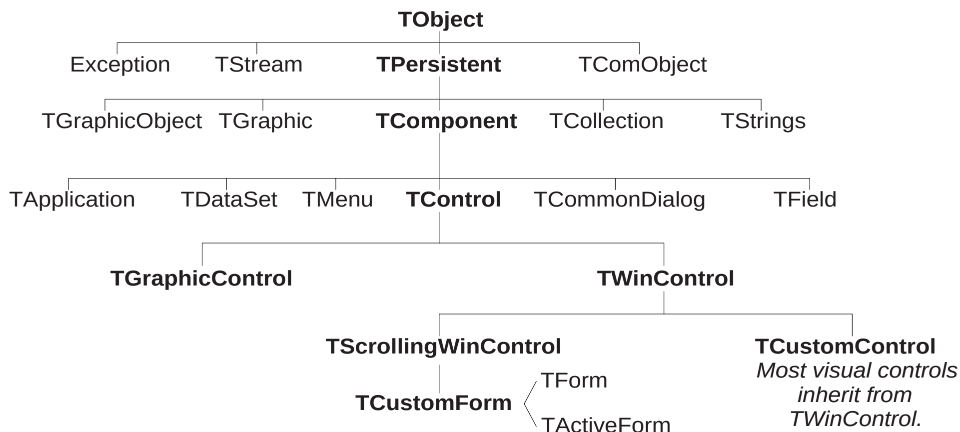
Per informazioni sull'installazione di nuovi componenti, consultare "Installazione di package di componenti" a pagina 15-6.

Librerie di classi

I componenti di C++Builder fanno tutti parte di due gerarchie di classi note come Visual Component Library (VCL) e Component Library for Cross Platform (CLX). La Figura 45.1 mostra la relazione delle classi selezionate che compongono la VCL. La gerarchia della CLX è simile a quella della VCL ma i controlli Windows sono chiamati widget (quindi *TWinControl* è chiamato TwidgetControl, ad esempio) ed esistono altre differenze. Per maggiori informazioni sulla gerarchia delle classi e delle relazioni di ereditarietà che intercorrono tra le classi, consultare il Chapter 46, "Programmazione object-oriented per gli scrittori di componenti". Per una panoramica delle differenze tra la CLX e la VCL, consultare "CLX e VCL" a pagina 14-5 per dettagli sui componenti consultare la Guida di riferimento in linea della CLX.

La classe *TComponent* è l'antenato comune di ogni componente nella VCL e nalla CLX. *TComponent* fornisce le proprietà e gli eventi minimi necessari a un componente per funzionare in C++Builder. I vari settori della libreria forniscono altre capacità più specifiche.

Figura 45.1 La gerarchia di classi della Visual Component Library



Quando si crea un componente, esso viene aggiunto alla VCL o alla CLX, derivando una nuova classe da uno dei tipi di classe già esistenti nella gerarchia.

Componenti e classi

Dal momento che i componenti sono classi, lo scrittore di componenti lavora con gli oggetti a un livello differente da quello dello sviluppatore di applicazioni. La creazione di nuovi componenti richiede che si derivino nuove classi.

In sintesi, esistono due differenze principali tra la creazione di componenti e il loro utilizzo nelle applicazioni. Nella creazione di componenti:

- Si ha accesso a parti della classe inaccessibili a chi sviluppa applicazioni.
- Si aggiungono nuove parti al componente (ad esempio, le proprietà).

Date queste differenze, chi scrive componenti deve prestare attenzione a un maggior numero di convenzioni e pensare attentamente a come i componenti da lui sviluppati verranno utilizzati da chi scrive applicazioni.

Creazione di componenti

Qualsiasi elemento di programma che si intende gestire in fase di progetto può diventare un componente. Creare un componente significa derivare una nuova classe da una esistente. Si può derivare un nuovo componente in molti modi:

- Modifica di controlli esistenti
- Creazione di controlli con finestra
- Creazione di controlli grafici
- Derivazioni di sottoclassi da controlli Windows
- Creazione di componenti non visuali

La Tabella 45.1 riepiloga i vari tipi di componenti e le classi che si possono usare come punto di partenza per ciascuna di queste tecniche.

Tabella 45.1 Punti di partenza per la creazione di componenti

Per raggiungere questo risultato	Cominciare con questo tipo
Modifica di un componente esistente	Qualsiasi componente esistente, come *TButton* o *TListBox*, oppure un tipo astratto di componente, come *TCustomListBox*
Creazione di un controllo con finestra (o basato su widget in CLX)	*TWinControl* (*TWidgetControl* in CLX)
Creazione di un controllo grafico	*TGraphicControl*
Creazione di una sottoclasse da un controllo	Qualsiasi controllo Windows (VCL) o basato su widget (CLX)
Creazione di un componente non visuale	*TComponent*

Si possono anche derivare classi che non sono componenti e che non possono essere gestite su una scheda, come *TRegIniFile* e *TFont*.

Modifica di controlli esistenti

Il modo più semplice per creare un componente consiste nel personalizzarne uno esistente. Si può derivare un nuovo componente da un qualsiasi altro fornito con C++Builder.

Alcuni controlli, come le caselle di riepilogo e le griglie, presentano diverse varianti di una stessa impostazione di base. In questi casi, la VCL e la CLX includono una classe astratta (il cui nome contiene la parola “Custom”, ad esempio *TCustomGrid*) da cui derivare versioni personalizzate.

Ad esempio, si potrebbe creare una casella di riepilogo speciale, priva di alcune proprietà della classe standard *TListBox*. Non è possibile rimuovere (nascondere) una proprietà ereditata da una classe antenato, quindi occorre derivare il componente da un oggetto di livello superiore rispetto a *TListBox* nella gerarchia. Anziché partire necessariamente dalla classe astratta *TWinControl* (o da *TWidgetControl* nella CLX) e reinventare tutte le funzioni della casella di riepilogo, la VCL mette a disposizione *TCustomListBox*, che implementa le proprietà della casella di riepilogo ma non le pubblica. Quando si deriva un componente da una classe astratta come *TCustomListBox*, si rendono pubbliche solo le proprietà che si vogliono rendere disponibili nel componente, lasciando protetto tutto il resto.

Il Capitolo 47, “Creazione di proprietà”, spiega come rendere pubbliche le proprietà ereditate. Il Capitolo 53, “Modifica di un componente esistente”, e il Capitolo 55,

"Personalizzazione di una griglia", mostrano alcuni esempi di come modificare i controlli esistenti.

Creazione di controlli con finestra

I controlli con finestra nella VCL e nella CLX sono oggetti che appaiono in esecuzione e con cui l'utente può interagire. Ogni controllo con finestra ha un handle di finestra, accessibile tramite la proprietà *Handle*, che consente al sistema operativo di identificare il controllo e di operare su di esso. Se si utilizzano controlli VCL, l'handle consente al controllo di ricevere il fuoco di input e può essere passato alle funzioni API di Windows. In CLX, questi controlli sono controlli basati su widget. Ogni controllo basato su widget ha un handle, accessibile tramite la proprietà *Handle*, che identifica il widget sottostante.

Tutti i controlli con finestra discendono dalla classe *TWinControl* (*TWidgetControl* in CLX). Fra questi vi sono numerosi controlli con finestra standard, quali pulsanti, caselle di riepilogo e caselle di testo. Benché sia possibile derivare direttamente da *TWinControl* (da *TWidgetControl* in CLX), un controllo originale (un controllo che non è in relazione con nessun altro controllo esistente), C++Builder fornisce appositamente per questo scopo il componente *TCustomControl*. *TCustomControl* è un controllo con finestra specializzato che semplifica il disegno di complesse immagini visuali.

Il Capitolo 55, "Personalizzazione di una griglia", presenta un esempio di creazione di un controllo con finestra.

Creazione di controlli grafici

Se non è necessario che il controllo riceva il fuoco di input, è possibile trasformarlo in un controllo grafico. I controlli grafici sono analoghi ai controlli per finestra, ma non hanno gli handle di finestra, e pertanto consumano meno risorse di sistema. Componenti come *TLabel*, che non ricevono il fuoco di input, sono controlli grafici. Anche se questi controlli non possono ricevere il fuoco, è possibile progettarli in modo che rispondano ai messaggi del mouse.

C++Builder supporta la creazione di controlli personalizzati grazie al componente *TGraphicControl*. *TGraphicControl* è una classe astratta derivata da *TControl*. Anche se è possibile derivare i controlli direttamente da *TControl*, è meglio iniziare da *TGraphicControl*, il quale fornisce un canvas su cui disegnare e, in Windows, è in grado di gestire i messaggi *WM_PAINT*; tutto ciò che occorre fare è ridefinire il metodo *Paint*.

Il Capitolo 54, "Creazione di un controllo grafico", contiene un esempio di creazione di un controllo grafico.

Derivazioni di sottoclassi da controlli Windows

Nella programmazione tradizionale in Windows i controlli personalizzati vengono creati definendo una nuova *classe window* e registrandola con Windows. La classe

window (simile agli *oggetti* o alle *classi* nella programmazione orientata agli oggetti) contiene informazioni condivise tra istanze dello stesso tipo di controllo; è possibile basare una nuova classe *window* su una esistente; questa operazione viene detta derivazione di sottoclassi (*subclassing*). Si inserisce quindi il controllo in una DLL (Dynamic Link Library), in modo analogo a quanto si fa per i controlli standard di Windows, e gli si fornisce un'interfaccia.

C++Builder consente di creare un "contenitore" di componenti per qualsiasi classe *window* esistente. Pertanto se si desidera usare nelle applicazioni C++Builder una libreria esistente di controlli personalizzati, è possibile creare componenti di C++Builder che si comportano come i propri controlli, e derivare i nuovi controlli esattamente come si farebbe con qualsiasi altro componente.

Per esempi delle tecniche usate nei controlli Windows di derivazione delle classi, vedere i componenti nel file header StdCtls che rappresentano i controlli Windows standard, come *TEdit*. Per esempi della CLX, vedere QStdCtls.

Creazione di componenti non visuali

I componenti non visuali sono utilizzati come interfacce per elementi come database (*TDataSet* o *TSQLConnection*) e clock di sistema (*TTimer*), e come segnaposti per le finestre di dialogo (*TCommonDialog* nella VCL o *TDialog* nella CLX e i relativi discendenti). È probabile che quasi tutti i componenti che si scrivono siano controlli visuali. I componenti non visuali possono essere derivati direttamente da *TComponent*, la classe base astratta di tutti i componenti.

Cosa mettere in un componente

Per far sì che i propri componenti siano parti affidabili dell'ambiente C++Builder, è necessario seguire determinate convenzioni durante la loro progettazione. Questa sezione affronta i seguenti argomenti:

- Rimozione delle dipendenze
- Impostazione di proprietà, metodi ed eventi
- Incapsulamento della grafica
- Registrazione di componenti

Rimozione delle dipendenze

Una qualità che rende i componenti utilizzabili è la totale assenza nel codice di restrizioni su ciò che possono fare. Per caratteristica intrinseca, i componenti sono incorporati all'interno delle applicazioni in combinazioni, ordini e contesti svariati. Si dovrebbero progettare componenti che funzionino in qualsiasi situazione, senza alcun prerequisito.

Un esempio eccellente di rimozione delle dipendenze è la proprietà *Handle* di *TWinControl*. Chi ha già scritto applicazioni Windows sa che uno degli aspetti più difficili e suscettibili di errori nell'esecuzione di un programma è di garantirsi,

chiamando la funzione API *CreateWindow*, da un tentativo di accesso a una finestra o a un controllo finché questi non vengono creati. I controlli per finestra di C++Builder evitano all'utente questa preoccupazione assicurando la disponibilità di un handle di finestra valido in caso di necessità. Il controllo può verificare se la finestra è stata creata utilizzando una proprietà che rappresenta l'handle della finestra; se l'handle non è valido, il controllo crea una finestra e restituisce l'handle. In questo modo, ogni volta che il codice di un'applicazione accede alla proprietà *Handle*, si ha la sicurezza di ottenere un handle valido.

Eliminando operazioni di base come la creazione di finestre, i componenti di C++Builder consentono agli sviluppatori di concentrarsi su ciò che vogliono realmente fare. Non c'è alcun bisogno di verificare se l'handle esiste o di creare la finestra, prima di passare un handle di finestra a una funzione API. Lo sviluppatore di applicazioni può dare per scontato il normale funzionamento, invece di continuare a verificare tutte le cose che possono provocare di errori.

Per quanto la creazione di componenti liberi da dipendenze richieda molto tempo, di solito è tempo ben speso. Non solo risparmia agli sviluppatori di applicazioni compiti ripetitivi e faticosi, ma riduce la quantità di documentazione e le incombenze del supporto.

Impostazione di proprietà, metodi ed eventi

Altre all'immagine visibile che può essere gestita in Form designer, gli attributi più ovvi di un componente sono le proprietà, gli eventi e i metodi. A ciascuno di questi argomenti è dedicato un capitolo; la spiegazione che segue mette in evidenza alcune motivazioni che ne giustificano l'utilizzo.

Proprietà

Le proprietà danno allo sviluppatore di applicazioni l'impressione di impostare o di leggere il valore di una variabile, mentre consentono allo scrittore di componenti di nascondere la struttura dati associata o di implementare particolari elaborazioni quando si accede al valore.

Ci sono molti vantaggi nell'utilizzo delle proprietà:

- Le proprietà sono disponibili in fase di progetto. Chi sviluppa applicazioni può impostare o modificare i valori iniziali delle proprietà senza dover scrivere una riga di programma.
- Le proprietà sono in grado di controllare i valori o i formati non appena vengono assegnati dallo sviluppatore di applicazioni. La convalida dell'input in fase di progettazione consente di prevenire errori.
- Il componente può creare su richiesta valori appropriati. L'errore di programmazione forse più frequente è quello di fare riferimento a una variabile non inizializzata. Rappresentando i dati con una proprietà, è possibile garantire che un valore sia sempre disponibile a richiesta.

- Le proprietà consentono di nascondere i dati con un'interfaccia semplice e coerente. È possibile modificare la struttura delle informazioni di una proprietà senza rendere percepibile il cambiamento a chi sviluppa applicazioni.

Il Capitolo 47, "Creazione di proprietà", spiega come aggiungere proprietà ai componenti.

Eventi

Un evento è una particolare proprietà che chiama del codice in risposta ad un input o ad altra attività in fase di esecuzione. Gli eventi offrono agli sviluppatori di applicazioni un modo per collegare blocchi specifici di codice a occorrenze specifiche in fase di esecuzione, come le azioni e la pressione dei tasti del mouse. Il codice che viene eseguito al verificarsi di un evento è detto *gestore di evento*.

Gli eventi consentono a chi sviluppa applicazioni di specificare le risposte per i diversi tipi di input, senza dover definire nuovi componenti.

Il Chapter 48, "Creazione di eventi", spiega come implementare gli eventi standard e come definirne di nuovi.

Metodi

I metodi di una classe sono funzioni che operano su una classe piuttosto che su specifiche istanze della classe. Per esempio, ogni metodo costruttore di un componente è un metodo della classe. I metodi dei componenti sono funzioni che operano sulle stesse istanze dei componenti. Chi sviluppa applicazioni usa i metodi per ordinare al componente di compiere un'azione specifica o di restituire un valore non contenuto in alcuna proprietà.

Poiché richiedono l'esecuzione di codice, i metodi possono essere chiamati soltanto in fase di esecuzione. I metodi risultano utili per svariati motivi:

- I metodi incapsulano le funzionalità di un componente nello stesso oggetto in cui risiedono i dati.
- I metodi possono nascondere procedure complesse con un'interfaccia semplice e coerente. Lo sviluppatore di applicazioni può chiamare un metodo *AlignControls* di un componente senza sapere come funziona o quali differenze presenta rispetto al metodo *AlignControls* di un altro componente.
- I metodi consentono di aggiornare diverse proprietà con una sola chiamata.

Il Capitolo 49, "Creazione di metodi", spiega come aggiungere metodi ai componenti.

Incapsulamento della grafica

C++Builder semplifica la gestione della grafica in Windows poiché incapsula i vari strumenti grafici in un canvas. Il canvas rappresenta la superficie di disegno di una finestra o di un controllo e contiene altre classi, come una penna, un pennello o un font. Un canvas è analogo a un contesto di dispositivo di Windows, ma effettua automaticamente la maggior parte della manutenzione.

Se si è già scritta un'applicazione grafica per Windows, si ha familiarità con i requisiti imposti dall'interfaccia GDI (Graphics Device Interface) di Windows. Per esempio, la GDI limita il numero di contesti disponibili per i dispositivi e richiede che gli oggetti grafici vengano ripristinati al loro stato iniziale prima di distruggerli.

C++Builder evita tutte queste preoccupazioni. Per disegnare su una scheda o su altro componente, si accede alla proprietà *Canvas* del componente. Se si desidera personalizzare una penna o un pennello, se ne impostano il colore o lo stile. Al termine, C++Builder si sbarazza delle risorse. C++Builder inoltre mette in cache le risorse per evitare di ricrearle nel caso l'applicazione utilizzi spesso lo stesso tipo di risorse.

È sempre possibile accedere liberamente alla GDI di Windows, ma non si può non constatare che, se si utilizza il canvas incluso nei componenti di C++Builder, spesso il codice è più semplice e viene eseguito più velocemente. Le caratteristiche grafiche sono spiegate in dettaglio nel Capitolo 50, "Uso della grafica nei componenti".

L'incapsulamento della grafica della CLX funziona diversamente. Un canvas è invece un painter. Per disegnare su una scheda o su altro componente, si accede alla proprietà *Canvas* del componente. *Canvas* è una proprietà ed è anche un oggetto di nome *TCanvas*. *TCanvas* funge da wrapper per un oggetto Qt painter a cui è possibile accedere attraverso la proprietà *Handle*. È possibile utilizzare lo handle per accedere a funzioni a basso livello delle libreria grafiche di Qt.

Se si desidera personalizzare una penna o un pennello, se ne impostano il colore o lo stile. Al termine, C++Builder si sbarazza delle risorse. CLX inoltre memorizza nella cache le risorse.

È possibile utilizzare il canvas incorporato nei componenti CLX derivandolo da essi. Il funzionamento delle immagini grafiche dipende dal canvas dell'oggetto da cui il componente deriva.

Registrazione di componenti

Prima di installare i propri componenti nell'IDE di C++Builder, è necessario registrarli. La registrazione comunica a C++Builder dove collocare il componente sulla Component palette. È anche possibile personalizzare il modo in cui C++Builder registra i componenti nel file di scheda. Per maggiori informazioni sulla registrazione di un componente, consultare il Capitolo 52, "Come rendere disponibili i componenti in fase di progetto".

Creazione di un nuovo componente

Per creare un nuovo componente si può procedere in due modi:

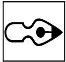
- Creazione di un componente con Component wizard
- Creazione manuale di un componente

È possibile usare uno di questi due metodi per creare un componente minimamente funzionale, pronto per essere installato nella Component palette. Dopo

l'installazione, si può aggiungere il nuovo componente ad una scheda e collaudarlo sia in fase di progetto sia in fase di esecuzione. In seguito sarà possibile aggiungere altre caratteristiche al componente, aggiornare la Component palette e continuare il collaudo.

Ci sono diverse operazioni fondamentali da eseguire ogni qualvolta si crea un nuovo componente. Questi aspetti vengono descritti di seguito; gli altri esempi riportati in questo documento presuppongono una sufficiente padronanza degli argomenti trattati.

1 Creare una unit per il nuovo componente.
2 Derivare il componente da un altro tipo di componente esistente.
3 Aggiungere proprietà, metodi e eventi.
4 Registrare il componente con C++Builder.
5 Creare un file di Guida per il componente e le relative proprietà, metodi ed eventi.



La creazione di un file di guida per fornire istruzioni agli utenti sull'uso del componente è facoltativa.

6 Creare un package (una particolare libreria a collegamento dinamico) in modo che sia possibile installare il componente nell'IDE di C++Builder.

Al termine, il componente completo include i seguenti file:

- Un file package (.BPL) o package collection (.PCE)
- Una libreria (.LIB) per il package
- Un file .BPI (Borland Import Library) per il package
- Un file unit compilato (.OBJ)
- Un file risorse compilato (.RES) per la mappa della palette
- Un file di guida (.HLP)

È anche possibile creare una bitmap per rappresentare il nuovo componente. Vedere "Creazione di una bitmap per un componente" a pagina 45-15.

I rimanenti capitoli della Parte V illustrano i diversi aspetti della costruzione dei componenti e forniscono numerosi esempi completi di scrittura di differenti tipi di componenti.

Creazione di un componente con Component wizard

Component wizard semplifica le fasi iniziali della creazione di un componente. Quando si utilizza Component wizard, è necessario specificare solo quanto segue:

- La classe da cui il componente è derivato.
- Il nome della classe del nuovo componente.
- La pagina della Component palette in cui si desidera che appaia.
- nome della unit in cui il componente verrà creato.
- Il percorso di ricerca per trovare la unit.
- Il nome del package in cui si desidera collocare il componente.

Component wizard esegue le stesse operazioni che si compiono creando un componente in modo manuale:

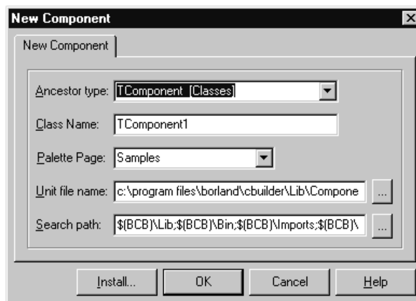
- Creazione di una unit (un file .CPP e il file header associato).
- Derivazione del componente.
- Dichiarazione di un nuovo costruttore.
- Registrazione del componente.

Component wizard non consente di aggiungere componenti a una unit esistente (costituita da un file .CPP e dal file header associato). Se si vogliono aggiungere nuovi componenti alla unit, occorre farlo manualmente.

1 Per avviare Component wizard, scegliere uno di questi metodi:

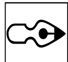
- Scegliere Component | New Component.
- Scegliere File | New | Other e fare doppio clic su Component.

Figura 45.2 Component wizard



Compilare i campi nel Component wizard:

2 Specificare nel campo Ancestor Type la classe da cui si sta derivando il nuovo componente.



Nella lista a discesa, molti componenti sono elencati due volte con diversi nomi di unit, con uno per la VCL e uno per la CLX. Le unit specifiche della CLX iniziano con Q (come QGraphics invece di Graphics). Accertartarsi di effettuare la derivazione dal componente corretto.

3 Specificare nel campo Class Name il nome della classe del nuovo componente.

4 Specificare nel campo Palette Page la pagina della Component palette in cui installare il nuovo componente.

5 Nel campo Unit file name, specificare il nome della unit in cui si vuole dichiarare la classe del componente.

6 Se la unit non si trova nel percorso di ricerca, modificare il percorso di ricerca nel campo Search Path.

7 Per collocare il componente in un nuovo package o in uno già esistente, fare clic su Component | Install e usare la finestra di dialogo visualizzata per specificare un package.



Se si deriva un componente da una classe VCL o CLX il cui nome inizia con “custom” (come *TCustomControl*), è bene evitare di collocarlo su una scheda finché non sono stati ridefiniti tutti i metodi astratti nel componente originale.

C++Builder non può creare oggetti istanza di una classe che ha proprietà o metodi astratti.

8 Dopo aver completato i campi del Component wizard, scegliere OK. C++Builder crea una nuova unit formata da un file .cpp e dal relativo file header associato.

Il file .cpp viene visualizzato nel Code editor. Il file contiene un costruttore per il componente e la funzione *Register* che registra il componente, dicendo a C++Builder quale componente aggiungere alla libreria di componenti e su quale pagina della Component palette dovrebbe essere visualizzato. Il file contiene inoltre un'istruzione include che specifica il file header che è stato creato. Ad esempio:

```
#include <vcl.h>
#pragma hdrstop
#include "NewComponent.h"
#pragma package(smart_init);
//---------------------------------------------------------------------------
// ValidCtrCheck is used to assure that the components created do not have
// any pure virtual functions.
//

static inline void ValidCtrCheck(TNewComponent *)
{
        new TNewComponent(NULL);
}
//---------------------------------------------------------------------------
__fastcall TNewComponent::TNewComponent(TComponent* Owner)
   : TComponent(Owner)
{
}
//---------------------------------------------------------------------------
namespace Newcomponent
{
   void __fastcall PACKAGE Register()
   {
      TComponentClass classes[1] = {__classid(TNewComponent)};
      RegisterComponents("Samples", classes, 0); //In CLX use a different page than Samples
   }
}
```

Nel caso di applicazioni CLX, nomi e ubicazioni di alcuni file sono differenti. Ad esempio, `<vcl\controls.hpp>` in CLX diventa `<clx\qcontrols.hpp>`.

Per aprire il file header nel Code editor, collocare il cursore sul nome del file header, fare clic destro sul pulsante del mouse in modo da visualizzare il menu contestuale e scegliere nel menu il comando Open File at Cursor.

Il file header contiene la nuova dichiarazione, composta da una dichiarazione di un costruttore e da parecchie istruzioni **#include** per supportare la nuova classe. Ad esempio:

```
#ifndef NewComponentH
#define NewComponentH
//---------------------------------------------------------------------------
#include <SysUtils.hpp>
#include <Controls.hpp>
```

```
#include <Classes.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------------
class PACKAGE TNewComponent : public TComponent
{
private:
protected:
public:
   __fastcall TNewComponent(TComponent* Owner);
__published:
};
//---------------------------------------------------------------------------
#endif
```

Prima di continuare, salvare il file .cpp assegnandogli un nome significativo.

Creazione manuale di un componente

Il modo più semplice per creare un nuovo componente è usare il Component wizard. La stessa procedura può, comunque, essere eseguita anche manualmente.

Per creare un componente in modo manuale, si devono effettuare le seguenti operazioni:

1 Creazione di un unit file
2 Derivazione del componente
3 Dichiarazione di un nuovo costruttore
4 Registrazione del componente

Creazione di un unit file

Una unit di C++Builder è formata da una combinazione di un file .CPP e di un file .H che vengono compilati in un file .OBJ. C++Builder utilizza le unit per svariati scopi. Ogni scheda ha la propria unit e anche la maggior parte dei componenti (o gruppi logici di componenti) ha la propria unit.

Quando si crea un componente, occorre creare una nuova unit per il componente o aggiungere il nuovo componente ad una unit esistente.

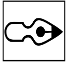
1 Per creare una unit per un componente, scegliere uno dei metodi seguenti:

   - Scegliere il comando File | New | Unit.
   - Scegliere il comando File | New | Other in modo da visualizzare la finestra di dialogo New Items, selezionare Unit e scegliere OK.

   C++Builder crea un file .CPP e un file header e visualizza il file .CPP nel Code editor. Salvare il file con un nome significativo.

2 Per aprire il file header, collocare nel Code editor il cursore sul nome del file header, fare clic destro e scegliere nel menu contestuale il comando Open File at Cursor.

3 Per aprire una unit esistente, selezionare File | Open e scegliere il codice sorgente della unit a cui si desidera aggiungere il componente.



Quando un componente viene aggiunto a una unit preesistente, è bene accertarsi che questa contenga il codice per i componenti. Per esempio, l'aggiunta del codice per un componente ad una unit che contiene una scheda provoca errori nella Component palette.

4. Una volta selezionata per il componente una unit nuova o preesistente, è possibile derivare la classe del componente.

Derivazione del componente

Ogni componente è una classe derivata da *TComponent*, da uno dei suoi discendenti più specializzati (come *TControl* o *TGraphicControl*) o da una classe di componente esistente. Il paragrafo“Creazione di componenti” a pagina 45-2 descrive da quale classe derivare i diversi tipi di componenti.

La derivazione di nuove classi viene illustrata in dettaglio nella sezione “Definizione di nuove classi” a pagina 46-1.

Per derivare una classe di componente, aggiungere una dichiarazione di classe al file header.

Una semplice classe di componenti è un componente non visuale derivato direttamente da *TComponent*.

Per creare una semplice classe di componenti, aggiungere la seguente dichiarazione di classe al file header:

```
class PACKAGE TNewComponent : public TComponent
{
};
```

La macro PACKAGE viene espansa in un'istruzione che consente di esportare e di importare le classi. Si dovrebbero inoltre aggiungere tutte le istruzioni include necessarie per specificare i file .HPP richiesti dal nuovo componente. Di seguito sono elencate le più comuni istruzioni include usate di solito:

```
#include <vcl\SysUtils.hpp>
#include <vcl\Controls.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Forms.hpp>
```

Finora il nuovo componente non fa nulla di diverso da *TComponent*. È stato solo creato un framework in cui costruire il nuovo componente.

Dichiarazione di un nuovo costruttore

Ogni nuovo componente deve avere un costruttore che ridefinisca il costruttore della classe da cui è stato derivato. Quando si scrive il costruttore del nuovo componente, si deve sempre chiamare il costruttore ereditato.

All'interno della dichiarazione della classe, dichiarare un costruttore virtuale nella sezione public della classe. Per maggiori informazioni sulla sezione public, consultare “Controllo dell'accesso” a pagina 46-4. Ad esempio:

```
class PACKAGE TNewComponent : public TComponent
{
public:
```

```
  virtual __fastcall TNewComponent(TComponent* AOwner);
};
```

Implementare il costruttore nel file .CPP:

```
__fastcall TNewComponent::TNewComponent(TComponent* AOwner): TComponent(AOwner)
{
}
```

All'interno del costruttore, va aggiunto il codice che si vuole far eseguire al momento della creazione del componente.

Registrazione del componente

La registrazione è un semplice procedimento che indica a C++Builder quali componenti aggiungere alla libreria di componenti e su quale pagina della Component palette dovranno apparire. Per una trattazione dettagliata del processo di registrazione, consultare il Capitolo 52, "Come rendere disponibili i componenti in fase di progetto".

Per registrare un componente:

1. Aggiungere al file .CPP della unit una funzione di nome *Register*, collocandola all'interno di un namespace. Il namespace è il nome del file in cui è incluso il componente, a cui viene sottratto l'estensione, da scrivere tutto in lettere minuscole ad eccezione della prima lettera.

   Ad esempio, il codice seguente esiste all'interno di un namespace *Newcomp*, dove *Newcomp* è il nome del file .CPP:

   ```
   namespace Newcomp
   {
     void __fastcall PACKAGE Register()
     {
     }
   }
   ```

2. All'interno della funzione *Register*, dichiarare un open array di tipo *TComponentClass* per contenere l'array dei componenti che si vogliono registrare. La sintassi dovrebbe essere simile a quella dell'esempio:

   ```
   TComponentClass classes[1] = {__classid(TNewComponent)};
   ```

   In questo caso, l'array delle classi contiene solo un componente, ma è possibile aggiungere all'array tutti i componenti che si vogliono registrare.

3. All'interno della funzione *Register*, chiamare *RegisterComponents* per ciascun componente che si vuole registrare.

   RegisterComponents è una funzione che accetta tre parametri: il nome di una pagina della Component palette, l'array delle classi dei componenti, e la dimensione – 1 delle classi dei componenti. Se si sta aggiungendo un componente ad una registrazione esistente, è possibile sia aggiungere il nuovo componente agli altri nell'istruzione che già esiste, sia aggiungere una nuova istruzione che chiama la funzione *RegisterComponents*.

Se tutti i componenti devono essere collocati sulla stessa pagina della Component palette, è possibile registrarli con una sola chiamata a *RegisterComponents*.

Per registrare un componente di nome *TNewComponent* e collocarlo sulla pagina Samples della Component palette, aggiungere al file .CPP della unit che dichiara *TNewComponent* la seguente funzione *Register*:

```
namespace Newcomp
{
  void __fastcall PACKAGE Register()
  {
    TComponentClass classes[1] = {__classid(TNewComponent)};
    RegisterComponents("Samples", classes, 0);
  }
}
```

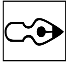
Questa chiamata a *Register* colloca *TNewComponent* sulla pagina Samples della Component palette.

Una volta registrato il componente, è possibile collaudarlo e infine installarlo nella Component palette. Questa procedura è descritta più dettagliatamente nella sezione "Installazione di un componente sulla Component palette" a pagina 45-20.

Creazione di una bitmap per un componente

Quando si crea un nuovo componente, è possibile definire delle bitmap personali per i componenti custom.

1 Scegliere Tools | Image Editor.
2 Nella finestra di dialogo Image Editor, scegliere File | New | Component Resource File (.dcr).
3 Nella finestra di dialogo untitled1.dcr, fare clic destro su Contents. Scegliere il comando New | Bitmap.
4 Nella finestra di dialogo Bitmaps Properties, modificare entrambe le proprietà Width ed Height a 24 pixel. Assicurarsi che sia selezionata l'opzione VGA (16 colors). Fare clic su OK.
5 Sotto Contents vengono visualizzati Bitmap e Bitmap1. Selezionare Bitmap1, fare clic destro e scegliere Rename. Assegnare alla bitmap lo stesso nome della classe del nuovo componente, inclusa la lettera T, utilizzando solo lettere in maiuscolo. Ad esempio, se il nuovo nome della classe sarà *TMyNewButton*, assegnare alla bitmap il nome TMYNEWBUTTON.



È necessario assegnare un nome usando solo lettere maiuscole, indipendentemente da come è scritto il nome assegnato alla classe nella finestra di dialogo New Component.



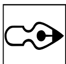
6 Fare doppio clic su TMYNEWBUTTON per visualizzare una finestra di dialogo con una bitmap vuota.

7 Utilizzare la tavolozza dei colori nella zono inferiore dell'Image Editor per progettare l'icona.

8 Scegliere il comando File | Save As e assegnare al file di risorse (.dcr o .res) lo stesso nome di base della unit in cui si vuole venga dichiarata la classe del componente. Ad esempio, assgnare al file di risorse il nome MyNewButton.dcr.

9 Scegliere il comando Component | New Component. Seguire le istruzioni a pagina 45-9. Assicurarsi che il sorgente del componente, MyNewButton.cpp, sia nella stessa directory di MyNewButton.dcr.

Per una classe di nome *TMyNewButton*, Component wizard assegna al sorgente del componente, o unit, il nome MyNewButton.cpp collocandolo, per impostazione predefinita, nella directory LIB. Fare clic sul pulsante Browse per trovare la nuova ubicazione della unit del componente appena generata.



Se per la bitmap si utilizza un file .res invece di un file .dcr, allora aggiungere un riferimento al sorgente del componente per collegare la risorsa. Ad esempio, se il nome del file .res è MyNewButton.res, dopo essersi assicurati che i file .cpp e .res sono nella stessa directory, aggiungono a MyNewButton.cpp quanto segue:

```
#pragma resource "*.res"
```

10 Scegliere il comando Component | Install Component per installare il componente in un package nuovo o esistente. Fare clic su OK.

Il nuovo package viene generato e poi installato. La bitmap che rappresenta il nuovo componente viene visualizzata sulla pagina della Component palette indicata nel Component wizard.

Verifica di componenti non installati

Prima di installarlo nella Component palette, si può verificare il comportamento in esecuzione del componente. Ciò è particolarmente utile per il debug di nuovi componenti appena creati, ma la stessa tecnica si applica a qualsiasi altro componente presente o meno nella Component palette. Per informazioni sulla verifica di componenti installati, consultare "Collaudo di componenti installati" a pagina 45-19.

La verifica di un componente senza installarlo offre il vantaggio di generare degli errori in compilazione che vengono visualizzati solo quando la classe viene istanziata. Ad esempio, se si cerca di creare un'istanza di una classe astratta, viene generato un errore che indica che occorre eseguire l'overload della pure virtual.

La verifica di un componente non installato viene seguita simulando le stesse azioni che esegue C++Builder quando si seleziona il componente nella tavolozza dei componenti e lo si colloca su una scheda.

Per verificare un componente non installato, procedere come segue:

1 Creare una nuova applicazione o aprirne una esistente.

2 Scegliere il comando Project | Add to Project per aggiungere al progetto la unit del componente.

3 Includere il file .H della unit del componente al file header della unit della scheda.

4 Aggiungere alla scheda un membro dati per rappresentare il componente.

   Questa è una delle principali differenze tra il modo in cui l'utente aggiunge i componenti e il modo in cui lo fa C++Builder. Nel primo caso il membro dati viene aggiunto alla sezione pubblica in fondo alla dichiarazione della classe della scheda. C++Builder lo aggiunge, invece, sopra, nella sezione published della dichiarazione della classe da lui gestita.

   Non si devono mai aggiungere i membri dati alla sezione della dichiarazione della classe della scheda gestita da C++Builder. Gli elementi di quella sezione della dichiarazione della classe corrispondono agli elementi memorizzati nel file della scheda. L'aggiunta sulla scheda di nomi di componenti che non esistono può invalidare il file della scheda.

5 Costruire il componente nel costruttore della scheda.

   Quando si chiama il costruttore del componente, occorre passare un parametro che specifichi il proprietario del componente (il componente responsabile della distruzione del componente al momento opportuno). Quasi sempre come proprietario viene passato **this**. In un metodo, **this** è un riferimento alla classe che contiene il metodo. In questo caso nel gestore *OnCreate* della scheda, **this** fa riferimento alla scheda.

6 Assegnare la proprietà *Parent*.

   L'impostazione della proprietà *Parent* è sempre la prima cosa da fare dopo aver costruito un controllo. Il genitore è il componente che contiene visualmente il controllo, che molto spesso è la scheda, ma potrebbe essere una casella o riquadro

di gruppo. Normalmente *Parent* viene impostato a **this**, cioè, alla scheda. *Parent* deve essere impostata sempre prima delle altre proprietà del controllo.

7 Impostare qualsiasi altra proprietà del componente nel modo desiderato.

Si supponga di voler verificare un componente della classe *TNewControl* in una unit denominata *NewCtrl*. Si deve creare un nuovo progetto, quindi seguire le diverse fasi per finire con un file header per la scheda simile a questo:

```
//---------------------------------------------------------------------------
#ifndef TestFormH
#define TestFormH
//---------------------------------------------------------------------------
#include <vcl\Classes.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Forms.hpp>
#include "NewCtrl.h"
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:          // IDE-managed Components
private:              // User declarations
public:               // User declarations
    TNewControl* NewControl1;
    __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

L'istruzione **#include** che include il file NEWCTRL.H parte dall'assunto che il componente risieda nella directory del progetto corrente o in una directory elencata nel percorso dei file include del progetto.

Questo è il file .CPP della unit della scheda:

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "TestForm.h"
#include "NewCtrl.h"
//---------------------------------------------------------------------------
#pragma package(smart_init);
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
static inline TNewControl *ValidCtrCheck()
{
    return new TNewControl(NULL);
}
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
    NewControl1 = new TNewControl(this);
```

```
    NewControl1->Parent = this;
    NewControl1->Left = 12;
}
//---------------------------------------------------------------------------
namespace Newctrl
{
    void __fastcall PACKAGE Register()
    {
        TComponentClass classes[1] = {__classid(TNewControl)};
        RegisterComponents("Samples", classes, 0);
    }
}
```

Collaudo di componenti installati

È possibile collaudare il comportamento in fase di progettazione di un componente dopo averlo installato nella Component palette. Ciò è particolarmente utile per il debug di nuovi componenti appena creati, ma la stessa tecnica si applica a qualsiasi altro componente presente o meno nella Component palette. Per informazioni su come collaudare i componenti che non sono stati ancora installati, consultare il paragrafo “Verifica di componenti non installati” a pagina 45-17.

Il collaudo di componenti dopo l’installazione consente di eseguire il debug dei componenti che generano eccezioni solamente in fase di progettazione, quando li si colloca su una scheda.

Collaudare un componente installato utilizzando una seconda istanza in esecuzione di C++Builder:

1 Dal menu dell’IDE di C++Builder selezionare il comando Project|Options e sulla pagina Directories/Conditional, impostare l’opzione Debug Source Path al file sorgente del componente.
2 Selezionare quindi il comando Tools|Debugger Options. Attivare le eccezioni che si desidera tracciare sulla pagina Language Exceptions.
3 Aprire il file sorgente del componente e impostare dei breakpoint.
4 Selezionare il comando Run|Parameters e impostare il campo Host Application al nome e all’ubicazione del file eseguibile di C++Builder.
5 Nella finestra di dialogo Run Parameters, fare clic sul pulsante Load per avviare la seconda istanza di C++Builder.
6 A questo punto, trascinare sulla scheda i componenti da collaudare, i quali si dovrebbero bloccare ai breakpoint fissati nel codice sorgente.

Installazione di un componente sulla Component palette

La procedura per aggiungere un componente alla Component palette è composta da due parti:

- Rendere disponibili i file sorgente.
- Aggiunta del componente.

Rendere disponibili i file sorgente

Tutti i file sorgente usati da un componente dovrebbero essere collocati nella stessa directory. Questi file includono i file di codice sorgente (.CPP e .PAS) e i file binari (.DFM, .RES, .RC e .DCR). I file header (.H e .HPP) dovrebbero essere collocati nella directory Include (o in una posizione raggiungibile mediante il percorso di ricerca globale dell'IDE o specifico del progetto).

Il processo di aggiunta di un componente comporta la creazione di un certo numero di file. Questi file vengono collocati automaticamente nelle directory specificate nelle opzioni dell'ambiente IDE (usare il comando di menu Tools | Environment Options e attivare la pagina Library). I file .LIB dovrebbero essere collocati nella directory di output specificata per i file BPI/LIB. Se l'aggiunta di un componente comporta la creazione di un nuovo package (in contrasto all'installazione in un package esistente), il file .BPL viene collocato nella directory di output relativa ai file BPL e i file .BPI nella directory di output BPI/LIB.

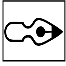
Aggiunta del componente

Per aggiungere i componenti alla libreria di componenti:

1. Scegliere il comando Component | Install Component.

   Verrà visualizzata la finestra di dialogo Install Component.

2. Decidere se installare il nuovo componente in un package esistente o in uno nuovo selezionando la pagina appropriata.
3. Immettere il nome del file .CPP contenente il nuovo componente oppure scegliere il pulsante Browse per trovare la unit.
4. Se il file .CPP non è nella posizione predefinita mostrata, modificare il percorso di ricerca del nuovo componente.
5. Immettere il nome del nuovo package in cui installare il componente oppure scegliere il pulsante Browse per trovare il package.
6. Se il componente viene installato in un nuovo package, se si vuole, è possibile immettere una descrizione significativa del package.
7. Scegliere OK per chiudere la finestra di dialogo Install Component. Così facendo il package viene compilato e generato e il componente installato sulla Component palette.



I componenti appena installati appaiono inizialmente sulla pagina della Component palette specificata da chi ha scritto il componente. Una volta installati nella tavolozza, è possibile spostare i componenti su una pagina differente utilizzando la finestra di dialogo Component|Configure palette.

Capitolo

46

Programmazione object-oriented per gli scrittori di componenti

Chi ha già scritto applicazioni con C++Builder, sa che una classe contiene sia dati sia codice, e che è possibile manipolare le classi sia in progettazione sia in esecuzione.

In questo senso, si è un utente di componenti.Quando si creano nuovi componenti, si opera con le classi con modalità che non saranno mai necessarie a chi sviluppa applicazioni. Si cerca anche di nascondere tutte le operazioni interne del componente agli sviluppatori che ne faranno uso. Scegliendo per il componente un antenato appropriato, progettando le interfacce in modo che espongano solo proprietà e metodi necessari allo sviluppatore e seguendo le altre direttive descritte in questo capitolo, si potranno creare versatili componenti riutilizzabili.

Prima di iniziare a creare componenti, si dovrebbe già avere familiarità con gli argomenti, relativi alla programmazione orientata agli oggetti (OOP), elencati di seguito):

- Definizione di nuove classi
- Antenati, discendenti e gerarchie della classe
- Controllo dell'accesso
- Dispatch dei metodi
- Membri astratti di classi
- Classi e puntatori

Definizione di nuove classi

La differenza tra chi scrive componenti e chi sviluppa applicazioni è che lo scrittore di componenti crea nuove classi mentre lo sviluppatore di applicazioni tratta istanze di classi.

Una classe è essenzialmente un tipo. Il programmatore lavora sempre con tipi e con istanze, anche se non utilizza questa terminologia. Crea, ad esempio, variabili di un

tipo, come gli *int*. Le classi sono di solito più complesse dei tipi semplici di dati, ma il loro modo di operare è lo stesso: l'assegnazione di valori differenti a istanze dello stesso tipo permette di eseguire compiti diversi.

Ad esempio, è piuttosto comune creare una scheda che contiene due pulsanti, uno identificato con OK e uno con Annulla. Entrambi sono istanze della classe *TButton*, ma il loro diverso comportamento dipende dall'assegnazione di valori differenti alle proprietà *Caption* e di gestori diversi ai loro eventi *OnClick*.

Derivazione di nuove classi

Ci sono due ragioni per derivare una nuova classe:

- Modificare le impostazioni predefinite della classe per evitare ripetizioni
- Aggiungere nuove capacità a una classe

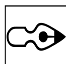
In entrambi i casi, il file è quello di creare oggetti riutilizzabili. Chi progetta componenti pensando al riutilizzo, può risparmiare in seguito molto lavoro. Si diano pure dei valori predefiniti utilizzabili, ma è bene renderli personalizzabili.

Modificare le impostazioni predefinite della classe per evitare ripetizioni

Molti programmatori tentano di evitare le ripetizioni. Chi scopre di riscrivere sempre le stesse righe di codice, può collocare il codice in una funzione o creare una libreria di routine utilizzabili in diversi programmi. Per i componenti si può fare lo stesso. Se ci si accorge che si variano sempre le stesse proprietà o si fanno le stesse chiamate ai metodi, è possibile creare un nuovo componente che faccia queste cose come impostazione predefinita.

Per esempio, si supponga che ogni volta che si crei un'applicazione, si aggiunga una finestra di dialogo per compiere una particolare operazione. Si può progettare una volta per tutte la finestra di dialogo, si possono impostarne le proprietà e si può installare nella Component palette un componente wrapper ad esso associato. Si può progettare una volta per tutte la finestra di dialogo, si possono impostarne le proprietà e si può installare nella Component palette un componente wrapper ad esso associato. Trasformando la finestra di dialogo in un componente riutilizzabile, non solo si elimina un lavoro ripetitivo, ma si incoraggia la standardizzazione e si riduce la probabilità di commettere errori ogni volta che si crea la finestra di dialogo.

Il Capitolo 53, "Modifica di un componente esistente", mostra un esempio di come modificare le proprietà predefinite di un componente.



Se si vogliono modificare solo le proprietà rese pubbliche di un componente esistente, oppure per salvare determinati gestori di evento di un componente o di un gruppo di componenti, potrebbe risultare conveniente creare un *modello di componente* che semplifica tali operazioni.

Aggiungere nuove capacità a una classe

Un motivo abbastanza comune che induce a creare nuovi componenti è l'aggiunta di capacità non presenti in componenti esistenti. Per fare ciò, si derivi il nuovo

componente da un componente esistente o da una classe base astratta, come *TComponent* o *TControl*.

È bene derivare il nuovo componente dalla classe che contiene il sottoinsieme di caratteristiche più simili a quelle volute. Si possono aggiungere capacità a una classe ma non toglierne; perciò se una classe di componente esistente contiene proprietà che *non* devono essere incluse nel nuovo componente, lo si dovrebbe derivare dall'antenato di quel componete.

Ad esempio, per aggiungere funzionalità a una casella di riepilogo, si potrebbe derivare il componente da *TListBox*. D'altra parte, per aggiungere nuove caratteristiche ma escludere alcune capacità standard della casella di riepilogo, è necessario derivare il componente da *TCustomListBox*, antenato di *TListBox*. In questo modo è possibile ricreare (o rendere visibili) solo le capacità desiderate della casella di riepilogo e aggiungere le nuove caratteristiche.

Il Capitolo 55, "Personalizzazione di una griglia", mostra un esempio di personalizzazione di una classe di componente astratta.

Dichiarazione di una nuova classe di componente

Oltre ai componenti standard, C++Builder mette a disposizione molte classi astratte progettate come basi per derivare nuovi componenti nuovi. La Tabella 45.1 a pagina 45-3 mostra le classi da cui è possibile partire per creare i propri componenti.

Per dichiarare una nuova classe di componenti aggiungere una dichiarazione di classe nel file header del componente.

Per esempio, questa è la dichiarazione di un semplice componente grafico:

```
class PACKAGE TSampleShape : public TGraphicControl
{
public:
    virtual __fastcall TSampleShape(TComponent* Owner);
};
```

Non dimenticare di includere la macro PACKAGE (definita in Sysmac.h) che consente di importare ed esportare le classi.

Una dichiarazione completa di componente include proprietà, membri dati e dichiarazioni dei metodi prima della parentesi graffa finale, ma anche una dichiarazione vuota è valida e rappresenta un punto di partenza per aggiungere le caratteristiche del componente.

Antenati, discendenti e gerarchie della classe

Chi sviluppa applicazioni parte dall'assunto che ogni controllo abbia proprietà di nome *Top* e *Left* che determinano la posizione del controllo sulla scheda. Non gli interessa che tutti i controlli ereditino queste proprietà da un antenato comune, nella fattispecie *TControl*. Nel creare un componente, tuttavia, occorre sapere da quale

classe derivarlo affinché erediti le caratteristiche adatte. Ed è bene sapere tutto ciò che il controllo eredita, per sfruttare tutte le caratteristiche ereditate senza doverle impostare di nuovo.

La classe da cui si deriva un componente è detta antenato immediato. Ogni componente eredita dal suo antenato immediato, dall'antenato dell'antenato e così via. Tutte delle classi da cui un componente eredita sono dette essere i suoi antenati; il componente è un *discendente* dei suoi antenati.

Tutte assieme, le relazioni antenato-discendente in un'applicazione costituiscono una gerarchia di classi. Ogni generazione nella gerarchia contiene molte più cose rispetto ai propri antenati, dal momento che ogni classe eredita tutto ciò che hanno gli antenati e aggiunge nuove proprietà e metodi o ridefinisce quelli esistenti.

Se non si specifica un antenato immediato, C++Builder deriva il componente dall'antenato predefinito, *TObject*. *TObject* è l'antenato di tutte le classi della gerarchia dell'oggetto.

La regola generale per scegliere l'oggetto da cui derivare un altro oggetto è semplice: si deve selezionare l'oggetto che contenga quanto più possibile di ciò che si vuole includere nel nuovo oggetto, ma che non includa alcunché di ciò che non si vuole nel nuovo oggetto. È sempre possibile aggiungere all'oggetto nuove cose, ma non è possibile toglierne.

Controllo dell'accesso

There Esistono cinque livelli di *controllo di accesso*-detto anche *visibilità*-su proprietà, metodi e membri dati. La visibilità determinano quale codice può accedere a quali parti della classe. Specificando la visibilità, si definisce l'*interfaccia* al componente.

La Tabella 46.1 mostra i livelli di visibilità, dal più restrittivo al più accessibile:

Tabella 46.1 Livelli di visibilità all'interno di un oggetto

Visibilità	Significato	Utilizzato per
private	Accessibile solo nella classe in cui è definito.	Occultamento dei dettagli implementativi.
protected	Accessibile solo nella classe in cui è definito e nei suoi discendenti.	Definizione dell'interfaccia per lo scrittore di componenti.
public	Accessibile a tutto il codice.	Definizione dell'interfaccia di esecuzione.
__automated	Accessibile a tutto il codice. Le informazioni di tipo automation vengono generate.	Solo OLE Automation.
__published	Accessibile a tutto il codice e dall'Object Inspector. Memorizzata in un file scheda.	Definizione dell'interfaccia di progettazione.

Occultamento dei dettagli implementativi

La dichiarazione di una parte di una classe come **private**, rende quella parte invisibile al codice esterno a meno che le funzioni non siano dichiarate friends della classe. Le parti **private** di una classe sono utili soprattutto per nascondere i dettagli implementativi agli utenti della classe. Poiché gli utenti della classe non possono accedere alle parti **private**, è possibile cambiare l'implementazione interna della classe senza alcuna influenza sul codice dell'utente.

Se non si specifica alcun controllo di accesso su un membro dati, su un metodo o su una proprietà, tale parte risulta **private**.

Di seguito è riportato un esempio, suddiviso in due parti, che illustra come il fatto di dichiarare **private** un membro dati impedisca agli utenti di accedere alle informazioni.

La prima parte è una unit di una scheda costituita da un file header e da un file .CPP che, nel gestore di evento *OnCreate* della scheda, assegna un valore a un membro dati privato. Poiché il gestore di evento viene dichiarato all'interno della classe *TSecretForm*, la unit viene compilata senza errori.

```
#ifndef HideInfoH
#define HideInfoH
//---------------------------------------------------------------------------
#include <vcl\SysUtils.hpp>
#include <vcl\Controls.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Forms.hpp>
//---------------------------------------------------------------------------
class PACKAGE TSecretForm : public TForm
{
__published:    // IDE-managed Components
    void __fastcall FormCreate(TObject *Sender);
private:
    int FSecretCode;                                 // declare a private data member
public:         // User declarations
  __fastcall TSecretForm(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TSecretForm *SecretForm;
//---------------------------------------------------------------------------
#endif
```

Il file .CPP è il seguente:

```
#include <vcl.h>
#pragma hdrstop
#include "hideInfo.h"
//---------------------------------------------------------------------------
#pragma package(smart_init);
#pragma resource "*.dfm"
TSecretForm *SecretForm;
//---------------------------------------------------------------------------
__fastcall TSecretForm::TSecretForm(TComponent* Owner)
  : TForm(Owner)
```

```
{
}
//---------------------------------------------------------------------------
void __fastcall TSecretForm::FormCreate(TObject *Sender)
{
  FSecretCode = 42;                             // this compiles correctly
}
```

La seconda parte dell'esempio è un'altra unit di scheda che tenta di assegnare un valore al membro dati *FSecretCode* della scheda *SecretForm*. Il file header della unit è il seguente:

```
#ifndef TestHideH
#define TestHideH
//---------------------------------------------------------------------------
#include <vcl\SysUtils.hpp>
#include <vcl\Controls.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Forms.hpp>
//---------------------------------------------------------------------------
class PACKAGE TTestForm : public TForm
{
__published:    // IDE-managed Components
    void __fastcall FormCreate(TObject *Sender);
public:         // User declarations
    __fastcall TTestForm(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern TTestForm *TestForm;
//---------------------------------------------------------------------------
#endif
```

Il file .CPP è riportato di seguito. Poiché il gestore di evento *OnCreate* tenta di assegnare un valore al membro dati private della scheda *SecretForm,* la compilazione non va a buon fine e viene visualizzato il messaggio di errore `'TSecretForm::FSecretCode' is not accessible.`

```
#include <vcl.h>
#pragma hdrstop
#include "testHide.h"
#include "hideInfo.h"
//---------------------------------------------------------------------------
#pragma package(smart_init);
#pragma resource "*.dfm"
TTestForm *TestForm;
//---------------------------------------------------------------------------
__fastcall TTestForm::TTestForm(TComponent* Owner)
  : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TTestForm::FormCreate(TObject *Sender)
{
  SecretForm->FSecretCode = 13;                 //compiler stops here with error message
```

```
}
```

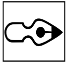
Anche se un programma che usa la unit *HideInfo* può usare classi di tipo *TSecretForm*, quello stesso programma non può accedere al membro dati *FSecretCode* di nessuna di quelle classi.

Nelle applicazioni CLX i nomi e le ubicazioni di alcuni file header sono differenti. Ad esempio, `<vcl\controls.hpp>` is `<clx\qcontrols.hpp>` in CLX.

Definizione dell'interfaccia per lo scrittore di componenti

La dichiarazione di una parte di una classe come **protected** rende quella parte visibile solamente alla stessa classe e ai suoi discendenti.

Si possono utilizzare le dichiarazioni **protected** per definire *l'interfaccia dello scrittore di componenti* alla classe. Le unit dell'applicazione non hanno accesso alle parti protette, ma ce l'hanno le classi derivate. Ciò significa che chi scrive componenti può modificare il modo in cui opera una classe senza renderne visibili i dettagli a chi sviluppa applicazioni.



Un errore comune è quello di provare ad accedere ai metodi protetti dall'interno di un gestore di evento. Di solito i gestori di evento sono metodi della scheda, e non del componente che riceve l'evento. Di conseguenza essi non hanno accesso ai metodi protetti del componente (a meno che il componente non sia dichiarato nella stessa unit della scheda).

Definizione dell'interfaccia di esecuzione

Se una parte di una classe viene dichiarata **public**, quella parte diventa visibile a ciascuna riga di codice che ha accesso alla classe.

Le parti pubbliche sono disponibile in esecuzione a tutto il codice, e pertanto le parti pubbliche di una classe definiscono la sua *interfaccia di esecuzione*. L'interfaccia di esecuzione è utile quegli elementi non significativi o inadatti in fase di progettazione, come le proprietà dipendenti da dati immessi in esecuzione o che sono a sola lettura. Devono essere resi pubblici anche i metodi che possono essere chiamati da chi sviluppa applicazioni.

Di seguito è riportato un esempio che mostra due proprietà a sola lettura dichiarate come parte dell'interfaccia di esecuzione di un componente:

```
class PACKAGE TSampleComponent : public TComponent
{
private:
    int FTempCelsius;                                  // implementation details are private
    int GetTempFahrenheit();
public:
    ⋮
    __property int TempCelsius = {read=FTempCelsius};          // properties are public
    __property int TempFahrenheit = {read=GetTempFahrenheit};
};
```

Questo è il metodo *GetTempFahrenheit* nel file .CPP:

```
int TSampleComponent::GetTempFahrenheit()
{
  return FTempCelsius * (9 / 5) + 32;
}
```

Definizione dell'interfaccia di progettazione

La dichiarazione di una parte di una classe come **published** rende quella parte pubblica e genera anche informazioni di tipo in esecuzione. Tra l'altro, le informazioni di tipo in , esecuzione consentono all' Object Inspector di accedere alle proprietà e agli eventi.

Poiché si possono vedere nell' Object Inspector, le parti pubblicate di una classe definiscono *l'interfaccia di progettazione* di quella classe. L'interfaccia di progettazione dovrebbe includere qualsiasi aspetto della classe che uno sviluppatore di applicazioni potrebbe voler personalizzare in fase di sviluppo, ma deve escludere qualsiasi proprietà dipendente da informazioni specifiche legate all'ambiente di esecuzione.

Le proprietà a sola lettura non possono essere parte dell'interfaccia di progettazione dato che chi sviluppa applicazioni non può assegnare loro dei valori. Le proprietà a sola lettura dovrebbero essere perciò public, invece che published.

Di seguito è riportato un esempio di una proprietà published di nome *Temperature*. Siccome è published, in fase di progettazione viene mostrata nell' Object Inspector.

```
class PACKAGE TSampleComponent : public TComponent
{
private:
    int FTemperature;
  ⋮
__published:
    __property int Temperature = {read=FTemperature, write=FTemperature};
};
```

Dispatch dei metodi

Dispatch è il termine usato per descrivere in che modo l'applicazione determina quali sono i metodi della classe che devono essere chiamati quando incontra una chiamata a un metodo della classe. Quando si scrive del codice che chiama un metodo della classe, la chiamata è simile ad una qualsiasi altra chiamata a funzione. Le classi, tuttavia, hanno due modi differenti di eseguire il dispatch dei metodi.

I due tipi di dispatch dei metodi si differenziano a seconda che siano per

- Metodi regolari (non virtuali)
- Metodi virtuali

Metodi regolari

I metodi di classe sono regolari (o non virtuali) a meno che non vengano dichiarati esplicitamente come virtuali, oppure a patto che essi ridefiniscano un metodo virtuale presente in una classe base. Il compilatore è in grado di determinare l'indirizzo esatto di un membro di una classe regolare in fase di compilazione. Questa funzionalità è nota come "binding in compilazione".

Un metodo regolare di una classe base viene ereditato dalle classi derivate. Nell'esempio seguente, un oggetto di tipo *Derived* può chiamare il metodo *Regular()* come se fosse un proprio metodo. La dichiarazione di un metodo in una classe derivata con lo stesso nome e gli stessi parametri di metodo regolare nella classe antenato sostituisce il metodo dell'antenato. Nell'esempio seguente, quando si chiama il metodo *d->AnotherRegular()*, esso viene smistato al metodo sostitutivo di *AnotherRegular()* della classe *Derived*.

```
class Base
{
public:
    void Regular();
    void AnotherRegular();
    virtual void Virtual();
};

class Derived : public Base
{
public:
    void AnotherRegular();          // replaces Base::AnotherRegular()
    void Virtual();                 // overrides Base::Virtual()
};

void FunctionOne()
{
  Derived *d;
  d = new Derived;
  d->Regular();                   // Calling Regular() as it were a member of Derived
                                  // The same as calling d->Base::Regular()
  d->AnotherRegular();            // Calling the redefined AnotherRegular(), ...
                                  // ... the replacement for Base::AnotherRegular()
  delete d;
}

void FunctionTwo(Base *b)
{
  b->Virtual();
  b->AnotherRegular();
}
```

Metodi virtuali

A differenza dei metodi regolari, che vengono collegati *in fase di compilazione*, i metodi virtuali vengono collegati in esecuzione. Il meccanismo virtuale di C++

consente di chiamare un metodo in base al tipo della classe che si sta utilizzando per chiamare il metodo.

Nell'esempio precedente, se si chiamasse la funzione *FunctionTwo()* con un puntatore a un oggetto di *Derived*, verrebbe chiamata la funzione *Derived::Virtual()*. Il meccanismo virtuale ispeziona dinamicamente in esecuzione il tipo della classe dell'oggetto che le viene passato e la smista al metodo opportuno. Ma la chiamata alla funzione regolare *b->AnotherRegular()* chiamerà sempre *Base::AnotherRegular()* in quanto l'indirizzo di *AnotherRegular()* è stato determinato in fase di compilazione.

Per dichiarare un nuovo metodo virtuale, anteporre alla dichiarazione del metodo la parola chiave **virtual**.

Quando il compilatore incontra la parola chiave **virtual**, crea un elemento nella tabella dei metodi virtuali della classe (VMT). La VMT conserva gli indirizzi di tutti i metodi virtuali presenti in una classe. Questa tabella di consultazione viene utilizzata in fase esecuzione per determinare che *b->Virtual* deve chiamare *Derived::Virtual()* invece di *Base::Virtual()*.

Quando si deriva una nuova classe da una classe preesistente, la nuova classe riceve una propria VMT, che comprende gli elementi della VMT dell'antenato a cui vengono aggiunti tutti i metodi virtuali dichiarati nella nuova classe. Inoltre, la classe derivata può ridefinire tutti i metodi virtuali ereditati.

Ridefinizione dei metodi

Ridefinire i metodi significa estendere o perfezionare un metodo di un antenato, invece che sostituirlo. Per ridefinire un metodo in una classe discendente, dichiarare di nuovo il metodo nella classe derivata, assicurandosi che il numero e il tipo degli argomenti siano gli stessi.

Il codice successivo mostra la dichiarazione di due semplici componenti. Il primo dichiara due metodi, ciascuno con un differente tipo di dispatch. L'altro, derivato dal primo, sostituisce il metodo non virtuale e ridefinisce il metodo virtuale.

```
class PACKAGE TFirstComponent : public TComponent
{
public:
  void Move();                            // regular method
  virtual void Flash();                   // virtual method
};

class PACKAGE TSecondComponent : public TFirstComponent
{
public:
  void Move();               // declares new method "hiding" TFirstComponent::Move()
  void Flash();              // overrides virtual TFirstComponent::Flash in TFirstComponent
};
```

Membri astratti di classi

Quando un metodo è dichiarato come **abstract** in una classe antenato, occorre renderlo evidente (dichiarandolo o implementandolo di nuovo) in qualsiasi

componente discendente prima di poter utilizzare nelle applicazioni il nuovo componente. C++Builder non è in grado di creare istanze di una classe che contiene membri astratti. Per maggiori informazioni sul modo di rendere evidenti le parti ereditate delle classi, consultare il Capitolo 47, "Creazione di proprietà", and Capitolo 49, "Creazione di metodi".

Classi e puntatori

Ogni classe (e perciò ogni componente) è in realtà un puntatore. Il fatto che la classe sia un puntatore diventa importante quando una classe viene passata come parametro. Come regola generale, si dovrebbero passare le classi per valore invece che per riferimento. Il motivo è che le classi sono già dei puntatori, e come tali sono riferimenti; passando una classe per riferimento equivale a passare un riferimento a un riferimento.

Capitolo

47

Creazione di proprietà

Le parti più evidenti dei componenti sono le proprietà. Chi sviluppa applicazioni può visualizzarle e manipolarle in fase di progettazione e avere un'immediata risposta di come il componente reagisce nel modulo di impostazione schede. Le proprietà ben progettate agevolano l'impiego del componente da parte di altri e ne assicurano una più facile manutenzione.

Per fare il miglior uso possibile delle proprietà dei componenti è necessario analizzare alcuni aspetti:

Perché creare le proprietà

Dal punto di vista dello sviluppatore di applicazioni, le proprietà appaiono come variabili. Gli sviluppatori possono impostarle o leggerne i valori come se fossero membri dati. (L'unica cosa che non si può fare con una proprietà, ma solo con una variabile, è passarla per riferimento a un metodo sotto forma di argomento.)

Le proprietà sono molto più potenti dei semplici membri dati perché

- Gli sviluppatori di applicazioni possono impostare le proprietà durante la progettazione. Mentre i metodi sono disponibili solo in fase di esecuzione, le proprietà consentono invece allo sviluppatore di personalizzare i componenti prima di eseguire l'applicazione. Le proprietà possono essere mostrate nell'Object Inspector, che semplifica il lavoro del programmatore: invece di gestire svariati parametri per costruire un oggetto, si lascerà a C++Builder il compito di leggerne i

valori dall'Object Inspector. L'Object Inspector convalida inoltre le assegnazioni delle proprietà non appena vengono eseguite.

- Le proprietà possono mascherare dettagli di implementazione. Ad esempio, i dati memorizzati internamente in formato crittografato possono apparire non crittografati come valori di una proprietà; anche se il valore è un semplice numero, il componente può cercare il valore in un database o eseguire calcoli complessi per determinarlo. Le proprietà consentono di aggiungere effetti complessi ad assegnazioni a prima vista semplici; ciò che appare un'assegnazione a un membro dati può essere invece una chiamata a un metodo che implica un'elaborazione complessa.
- Le proprietà possono essere virtuali. Quindi, ciò che a uno sviluppatore di applicazioni appare come una proprietà singola può essere invece implementato in modo diverso in componenti differenti.

Un semplice esempio è la proprietà *Top* di tutti i controlli.L'assegnazione di un nuovo valore a *Top* non solo modifica un valore memorizzato, ma riposiziona ed aggiorna il controllo. E gli effetti dell'impostazione di una proprietà non sono necessariamente limitati a un singolo componente. Per esempio, l'impostazione a **true** della proprietà *Down* di uno SpeedButton provoca l'impostazione a **false** di tutti gli altri pulsanti appartenenti allo stesso gruppo.

Tipi di proprietà

Una proprietà può essere di un qualunque tipo. Tipi diversi sono visualizzati in modo differente nell'Object Inspector, che, in fase di progetto, convalida le assegnazioni di proprietà non appena vengono create.

Tabella 47.1 Come appaiono le proprietà nell'Object Inspector

Tipo di proprietà	Trattamento dell' Object Inspector
Semplice	Le proprietà numeriche, di tipo carattere e stringa appaiono nell'Object Inspector rispettivamente come numeri, caratteri e stringhe. L'utente può scrivere e modificare direttamente il valore della proprietà.
Enumerato	Le proprietà dei tipi enumerati (compreso Boolean) appaiono come stringhe di testo modificabili. L'utente può far scorrere tutti i possibili valori facendo doppio clic nella colonna del valore. Appare anche un elenco a discesa che mostra tutti i valori possibili del tipo enumerato.
Set	Le proprietà dei tipi set appaiono nell'Object Inspector come un insieme. Espandendo l'insieme con un doppio clic del mouse, l'utente può trattare ogni elemento del set come un valore booleano: **true** se l'elemento è incluso nel set, **false** se non lo è).
Oggetto	Le proprietà che sono a loro volta classi spesso dispongono di propri editor, specificati nella procedura di registrazione del componente. Tuttavia, se la classe che fa capo a una proprietà contiene a sua volta la proprietà Published, l'Object Inspector permette all'utente di espandere l'elenco (con un doppio clic) per includere quelle proprietà e di modificarle individualmente. Le proprietà dell'oggetto devono discendere da *TPersistent*.

Tabella 47.1 Come appaiono le proprietà nell'Object Inspector (continua) (continua)

Tipo di proprietà	Trattamento dell' Object Inspector
Interface	Le proprietà che sono interfacce possono essere visualizzate nell'Object Inspector purché il valore sia un'interfaccia che è implementata da un componente (un discendente di *TComponent*). Le proprietà interfaccia hanno spesso i propri editor di proprietà.
Array	Le proprietà array devono avere i propri editor; l' Object Inspector non ha il supporto interno per la modifica delle proprietà array. È possibile specificare un editor di proprietà quando si registra il componente.

Pubblicazione di proprietà ereditate

Tutti i componenti ereditano proprietà dalle proprie classi antenate. Quando si deriva un componente da uno esistente, il nuovo componente eredita tutte le proprietà del suo antenato immediato. Se si deriva il componente da una classe astratta, la maggior parte delle proprietà ereditate sono protected (protette) o public (pubbliche) ma non sono published (rese pubbliche.

Per rendere disponibile in fase di progettazione nell'Object Inspector una proprietà protetta o pubblica bisogna ridichiararla come published. Ridichiarare significa aggiungere una dichiarazione per la proprietà ereditata alla dichiarazione della classe discendente.

Se si deriva un componente VCL da *TWinControl*, per esempio, esso eredita anche la proprietà *DockSite* protetta. Ridichiarando invece *DockSite* nel nuovo componente è possibile cambiare il livello di protezione e trasformarlo in public o published.

Il codice che segue mostra come ridichiarare *DockSite* in modo published rendendolo disponibile durante lo sviluppo.

```
class PACKAGE TSampleComponent : public TWinControl
{
__published:
    __property DockSite;
};
```

Nel ridichiarare una proprietà si specifica solo il nome della proprietà, non il tipo e le altre informazioni descritte nel paragrafo successivo “Definizione delle proprietà”. Si possono anche dichiarare nuovi valori predefiniti e specificare se memorizzare la proprietà.

La ridichiarazione può diminuire la protezione di una proprietà, ma non aumentarla. Pertanto, è possibile rendere pubblica una proprietà protetta, ma non è consentito nascondere una proprietà pubblica ridichiarandola come protetta.

Definizione delle proprietà

Questa sezione illustra come dichiarare nuove proprietà e spiega alcune convenzioni seguite per i componenti standard. Gli argomenti trattati sono i seguenti:

Dichiarazione della proprietà

AUna proprietà è dichiarata nella dichiarazione della classe del componente. Per dichiarare una proprietà, occorre specificare tre cose:

- Il nome della proprietà.
- Il tipo della proprietà.
- I metodi usati per leggere e scrivere il valore della proprietà. Se non viene dichiarato il metodo di scrittura, la proprietà è a sola lettura.

Le proprietà dichiarate in una sezione **__published** della dichiarazione della classe del componente sono modificabili in fase di progetto con Object Inspector. Il valore di una proprietà resa pubblica viene salvato assieme al componente nel file della scheda. Le proprietà dichiarate in una sezione **public** sono disponibili in fase di esecuzione e possono essere lette o impostate da programma.

Di seguito è riportata una tipica dichiarazione di proprietà di nome *Count*.

```
class PACKAGE TYourComponent : public TComponent
{
private:
    int FCount;                                              // data member for storage
    int __fastcall GetCount();                               // read method
    void __fastcall SetCount( int ACount );                  // write method
public:
    __property int Count = {read=GetCount, write=SetCount};  // property declaration
  ⋮
};
```

Memorizzazione interna dei dati

Non vi sono restrizioni su come memorizzare il dato di una proprietà. In generale, comunque, i componenti di C++Builder seguono queste convenzioni:

- I dati delle proprietà vengono memorizzati all'interno dei membri dati della classe.
- I membri dati utilizzati per memorizzare i dati della proprietà sono privati e dovrebbero essere accessibili soltanto dall'interno del componente stesso. I componenti derivati dovrebbero utilizzare la proprietà ereditata: non hanno bisogno di accedere direttamente all'area di memorizzazione interna dei dati della proprietà.

- I nomi degli identificatori di questi membri dati iniziano con la lettera *F* seguita dal nome della proprietà. Per esempio, i dati della proprietà *Width* definita in *TControl* sono memorizzati in un membro dati di nome *FWidth*.

Il principio che è alla base di queste convenzioni è che solo i metodi di implementazione di una proprietà dovrebbero avere accesso ai dati cui essa si riferisce. Se un metodo o un'altra proprietà ha bisogno di cambiare quel dato, dovrebbe farlo utilizzando la proprietà e non accedendo direttamente ai dati memorizzati. Ciò garantisce che l'implementazione di una proprietà può cambiare senza invalidare i componenti derivati.

Accesso diretto

L'accesso diretto è la maniera più semplice per rendere disponibili i dati delle proprietà. Ciò significa che le sezioni **read** e **write** della dichiarazione della proprietà specificano che l'assegnazione o la lettura del valore della proprietà accedono direttamente al membro dati di memorizzazione interna senza chiamare un metodo di accesso. L'accesso diretto è utile se si vuole rendere disponibile una proprietà nell'Object Inspector ma si vuole evitare che le modifiche al valore provochino un'elaborazione immediata.

È abbastanza comune avere accesso diretto per la sezione **read** di una dichiarazione di proprietà ma usare un metodo di accesso per la sezione **write**. Ciò fa sì che lo stato del componente venga aggiornato quando il valore della proprietà cambia.

La dichiarazione di un tipo componente riportata di seguito mostra una proprietà che utilizza l'accesso diretto sia per la parte **read** sia per la parte **write**.

```
class PACKAGE TSampleComponent : public TComponent
{
private:                                        // internal storage is private
    bool FReadOnly;                             // declare data member to hold value
  ⋮
__published:                                    // make property available at design time
    __property bool ReadOnly = {read=FReadOnly, write=FReadOnly};
};
```

Access methods

Nelle parti **read** e **write** di una dichiarazione di proprietà si può specificare, anziché un membro dati, un metodo di accesso. I metodi di accesso devono essere protetti e di solito sono dichiarati come **virtual**; ciò permette al componente discendente di ridefinire l'implementazione della proprietà.

Bisogna evitare di rendere pubblici i metodi di accesso. Tenerli protetti assicura che lo sviluppatore di applicazioni non modifichi inavvertitamente una proprietà chiamando un metodo.

Di seguito è riportata una classe che dichiara tre proprietà utilizzando lo specificatore index, il che consente alle tre proprietà di utilizzare gli stessi metodi di accesso in lettura e scrittura:

```
class PACKAGE TSampleCalendar : public TCustomGrid
{
private:
  int __fastcall GetDateElement(int Index);    // note Index parameter
  void __fastcall SetDateElement(int Index, int Value);
public:
  __property int Day = {read=GetDateElement, write=SetDateElement, index=3, nodefault};
  __property int Month = {read=GetDateElement, write=SetDateElement, index=2, nodefault};
  __property int Year = {read=GetDateElement, write=SetDateElement, index=1, nodefault};
};
```

Poiché ogni elemento della data (giorno, mese e anno) è un intero e poiché l'impostazione di ciascuno di essi richiede la codifica della data, il codice evita la duplicazione condividendo i metodi di lettura e scrittura di tutte e tre le proprietà. Occorre quindi solo un metodo per leggere un elemento della data e un altro metodo per scriverlo.

Di seguito è riportato il metodo di lettura per ottenere un elemento della data:

```
int __fastcall TSampleCalendar::GetDateElement(int Index)
{
  unsigned short AYear, AMonth, ADay;
  int result;
  FDate.DecodeDate(&AYear, &AMonth, &Aday); // break date into elements
  switch (Index)
  {
    case 1: result = AYear;  break;
    case 2: result = AMonth; break;
    case 3: result = ADay;   break;
    default: result = -1;
  }
  return result;
}
```

Questo è il metodo di scrittura usato per impostare l'opportuno elemento della data:

```
void __fastcall TSampleCalendar::SetDateElement(int Index, int Value)
{
  unsigned short AYear, AMonth, ADay;
  if (Value > 0) // all elements must be positive
  {
    FDate.DecodeDate(&AYear, &AMonth, &ADay); // get date elements
    switch (Index)
    {
      case 1: AYear = Value;  break;
      case 2: AMonth = Value; break;
      case 3: ADay = Value;   break;
      default: return;
    }
  }
  FDate = TDateTime(AYear, AMonth, ADay); // encode the modified date
  Refresh();// update the visible calendar
}
```

Il metodo read

Il metodo read per una proprietà è una funzione che non richiede parametri (tranne nei casi citati di seguito) e restituisce un valore dello stesso tipo della proprietà. Per convenzione il nome della funzione è *Get* seguito dal nome della proprietà. Per esempio, il metodo read relativo a una proprietà di nome *Count* sarà *GetCount*. Il metodo read manipola i dati memorizzati internamente in modo da produrre il valore della proprietà con il tipo appropriato.

Le uniche eccezioni alla regola che impone l'assenza di parametri sono le proprietà array e quelle che utilizzano specificatori di indice (vedere il paragrafo "Creazione di proprietà array" a pagina 47-8), in quanto entrambe passano i propri valori di indice come parametri. (Per creare un singolo metodo read condiviso da più proprietà, usare gli specificatori di indice.)

ISe non si dichiara un metodo read, la proprietà è a sola scrittura. Le proprietà a sola scrittura sono utilizzate raramente.

Il metodo write

Il metodo write per una proprietà è una funzione membro che richiede un solo parametro dello stesso tipo della proprietà (tranne nei casi citati di seguito). Il parametro può essere passato per riferimento o per valore e può avere un nome qualsiasi. Per convenzione, il nome del metodo write è *Set* seguito dal nome della proprietà. Ad esempio, il metodo write di una proprietà chiamata *Count* sarà *SetCount*. Il valore passato nel parametro diventa il nuovo valore della proprietà; il metodo write deve compiere tutte le operazioni necessarie per collocare il dato appropriato nell'area di memorizzazione interna della proprietà.

Le uniche eccezioni alla regola del singolo parametro sono le proprietà array e quelle che utilizzano specificatori di indice, in quanto passano i propri valori di indice come secondo parametro. (Per creare un singolo metodo write condiviso da più proprietà, usare gli specificatori di indice.)

Se non si dichiara un metodo write, la proprietà è a sola lettura.

I metodi write di solito verificano se un nuovo valore differisce dal valore attuale prima di modificare la proprietà. Per esempio, di seguito è riportato un semplice metodo write per una proprietà di tipo intero di nome *Count* che memorizza il suo valore attuale in un membro dati di nome *FCount*.

```
void __fastcall TMyComponent::SetCount( int Value )
{
  if ( Value != FCount )
  {
    FCount = Value;
    Update();
  }
```

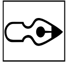
Valori predefiniti delle proprietà

Quando si dichiara una proprietà, è possibile specificare un *valore predefinito*. C++Builder utilizza il valore predefinito per determinare se memorizzare la

proprietà in un file di scheda. Se non si specifica un valore predefinito per una proprietà, C++Builder memorizza sempre la proprietà.

Per dichiarare un valore predefinito per una proprietà, aggiungere un segno di uguale dopo il nome della proprietà e impostare una coppia di parentesi graffe in cui includere la parola chiave **default** seguita dal valore predefinito. Ad esempio:

```
__property bool IsTrue = {default=true};
```



La dichiarazione di un valore predefinito non imposta la proprietà a quel valore. Il metodo costruttore del componente deve inizializzare i valori della proprietà quando è necessario. Comunque, poiché gli oggetti inizializzano sempre i propri membri dati a 0, non è strettamente necessario che il costruttore imposti le proprietà di tipo integer a 0, quelle di tipo stringa a null o quelle booleane a **false**.

Specifica di una proprietà senza valore predefinito

Quando una proprietà viene ridichiarata, è possibile specificare che non ha un valore predefinito, anche se la proprietà ereditata ne specificava uno.

Per designare una proprietà che non ha un valore predefinito, aggiungere un segno di uguale dopo il nome della proprietà e impostare una coppia di parentesi graffe in cui includere la parola chiave **nodefault**. Ad esempio:

```
__property int NewInteger = {nodefault};
```

Quando si dichiara una proprietà per la prima volta, non è necessario includere **nodefault**. L'assenza di un valore dichiarato predefinito significa che non c'è un valore predefinito.

Di seguito viene riportata la dichiarazione di un componente che include una singola proprietà booleana di nome *IsTrue* con un valore predefinito pari a **true**, e il costruttore che imposta il valore predefinito.

```
class PACKAGE TSampleComponent : public TComponent
{
private:
     bool FIsTrue;
public:
    virtual __fastcall TSampleComponent( TComponent* Owner );
__published:
    __property bool IsTrue = {read=FIsTrue, write=FIsTrue, default=true};
};

__fastcall TSampleComponent::TSampleComponent ( TComponent* Owner )
  : TComponent ( Owner )
{
  FIsTrue = true;
}
```

Creazione di proprietà array

Alcune proprietà si prestano a essere indicizzate come gli array. Per esempio, la proprietà *Lines* di *TMemo* è un elenco indicizzato delle stringhe che costituiscono il

testo del memo; si può trattarlo come un array di stringhe. *Lines* consente di accedere in modo naturale a un particolare elemento (una stringa) in una serie di dati più estesa (il testo del memo).

Le proprietà array vengono dichiarate come le altre proprietà, a meno che

- La dichiarazione comprenda uno o più indici che hanno un tipo specificato. Gli indici possono essere di qualsiasi tipo.
- Le parti **read** e **write** della dichiarazione di proprietà, se specificate, devono essere dei metodi. Non possono essere membri dati.

I metodi read e write per una proprietà array accettano parametri aggiuntivi che corrispondono agli indici. I parametri devono essere nello stesso ordine e dello stesso tipo degli indici specificati nella dichiarazione.

Ci sono alcune differenze importanti tra le proprietà array e gli array. A differenza dell'indice di un array, l'indice di una proprietà array non deve essere di tipo integer. Si può indicizzare una proprietà su una stringa, per esempio. Inoltre, si può far riferimento solo ai singoli elementi di una proprietà array e non all'intero intervallo della proprietà.

L'esempio seguente mostra la dichiarazione di una proprietà che restituisce una stringa basata su un indice integer.

```
class PACKAGE TDemoComponent : public TComponent
{
private:
    System::AnsiString __fastcall GetNumberSize(int Index);
public:
    __property System::AnsiString NumberSize[int Index] = {read=GetNumberSize};
  ⋮
};
```

Questo è il metodo *GetNumberSize* nel file .CPP:

```
System::AnsiString __fastcall TDemoComponent::GetNumberSize(int Index)
{
  System::AnsiString Result;
  switch (Index)
  {
    case 0:
      Result = "Zero";
      break;
    case 100:
      Result = "Medium";
      break;
    case 1000:
      Result = "Large";
      break;
    default: Result = "Unknown size";
  }
  return Result;
}
```

Creazione di proprietà per i sottocomponenti

Per impostazione predefinita, quando il valore di una proprietà è un altro componente, si assegna un valore a quella proprietà aggiungendo alla scheda o al modulo dati un'istanza dell'altro componente e quindi assegnando quel componente come valore della proprietà. Tuttavia, è anche possibile che il componente crei la propria istanza dell'oggetto che implementa il valore della proprietà. Un tale componente dedicato viene detto sottocomponente.

Tutti gli oggetto persistenti (tutti i discendenti di *TPersistent*) possono essere sottocomponenti. A differenza dei singoli componenti che, talvolta, possono essere assegnati come valore di una proprietà, le proprietà pubblicate dei sottocomponenti vengono salvate con il componente che li crea. Affinché ciò sia vero, comunque, devono essere soddisfatte le seguenti condizioni:

- L'*Owner* del sottocomponente deve essere il componente che lo crea e lo utilizza come valore di una proprietà pubblicata. Nel caso di sottocomponenti che discendono da *TComponent*, è possibile fare ciò impostando la proprietà *Owner* del sottocomponente. Per altri sottocomponenti, è necessario ridefinire il metodo *GetOwner* dell'oggetto persistente in modo che restituisca il componente che sta creando.
- Se il sottocomponente è un discendente di *TComponent*, deve indicare di essere un sottocomponente chiamando il metodo *SetSubComponent*. Di solito, questa chiamata viene effettuata o dal possessore al momento della creazione del sottocomponente oppure dal costruttore del sottocomponente.

Di solito, le proprietà i cui valori sono sottocomponenti sono a sola lettura. Se si permette di modificare una proprietà il cui valore è un sottocomponente, il setter della proprietà deve liberare il sottocomponente nel momento in cui al valore della proprietà viene assegnato un altro componente. Inoltre, il componente spesso crea una nuova istanza del proprio sottocomponente quando la proprietà viene impostata a NULL. In caso contrario, se la proprietà fosse stata modificata in un altro componente, il sottocomponente non potrebbe essere mai ripristinato in fase di progettazione. Il seguente esempio illustra un setter di proprietà per una proprietà il cui valore è un *TTimer*:

```
void __fastcall TDemoComponent::SetTimerProp(ExtCtrls::TTimer *Value)
{
  if (Value != FTimer)
  {
    if (Value)
    {
      if (FTimer && FTimer->Owner == this)
        delete FTimer;
      FTimer = Value;
      FTimer->FreeNotification(this);
    }
    else // NULL value
    {
      if (FTimer && FTimer->Owner != this)
      {
```

```
          FTimer = new ExtCtrls::TTimer(this);
          FTimer.SetSubComponent(true);
          FTimer->FreeNotification(this);
        }
      }
    }
  }
```

Si noti che il setter della proprietà visto in precedenza ha effettuato una chiamata al metodo *FreeNotification* del componente che è stato impostato come valore della proprietà. Questa chiamata garantisce che il componente che è stato assegnato come valore della proprietà invii una notifica nel caso debba essere distrutto. La notifica viene inviata chiamando il metodo *Notification*. Questa chiamata viene gestita ridefinendo il metodo *Notification* nel seguente modo:

```
void __fastcall TDemoComponent::Notification(Classes::TComponent *AComponent,
Classes::TOperation Operation)
{
  TComponent::Notification(AComponent, Operation); { call inherited method }
  if ((Operation == opRemove) && (AComponent == (TComponent *)FTimer))
    FTimer = NULL;
}
```

Registrazione e caricamento delle proprietà

C++Builder registra le schede e i loro componenti in file scheda (.dfm nella VCL and .xfm nella CLX). Un file scheda contiene le proprietà di una scheda e dei suoi componenti. Gli sviluppatori in C++Builder aggiungono nelle loro schede componenti scritti da altri; tali componenti devono avere la capacità di scrivere le loro proprietà nel file scheda quando vengono salvati. Analogamente, quando vengono caricati in C++Builder o vengono eseguiti come parte di un'applicazione, i componenti devono essere in grado di ripristinarsi automaticamente leggendo le proprie proprietà dal file scheda.

Generalmente non occorre fare nulla perché i componenti siano in grado di lavorare con i file scheda, in quanto la capacità di registrare e di ricaricare una rappresentazione fa parte del comportamento che essi hanno ereditato. Talvolta, comunque, può essere necessario modificare il modo in cui un componente si registra o il modo in cui si inizializza quando viene caricato; pertanto, è importante conoscere i meccanismi sottostanti.

Questi sono gli aspetti della registrazione delle proprietà che occorre conoscere:

- Uso del meccanismo di registrazione e di caricamento
- Specifica di valori predefiniti
- Determinazione di che cosa registrare
- Inizializzazione dopo il caricamento
- Registrazione e caricamento di proprietà non pubbliche

Uso del meccanismo di registrazione e di caricamento

La descrizione di una scheda è costituita dall'elenco delle sue proprietà e da descrizioni analoghe per ogni suo componente. Ogni componente, compresa la scheda stessa, è responsabile della registrazione e del caricamento della propria descrizione.

Per impostazione predefinita, quando un componente si registra, scrive i valori di tutte le sue proprietà published che differiscono dai valori predefiniti nell'ordine in cui sono state dichiarati. Quando un componente si carica automaticamente, come prima operazione si ricostruisce, impostando tutte le proprietà al loro valore predefinito, dopo di che legge i valori non predefiniti delle proprietà registrate.

Questo meccanismo predefinito viene utilizzato da molti componenti, e non richiede alcuna azione da parte dello scrittore di componenti. Ci sono, comunque, diversi metodi per personalizzare i processi di registrazione e di caricamento in modo da adattarli alle esigenze degli specifici componenti.

Specifica di valori predefiniti

I componenti C++Builder salvano i valori delle loro proprietà solo se differiscono da quelli predefiniti. C++Builder, se non si specifica diversamente, considera sempre una proprietà senza valore predefinito, il che comporta che il componente debba registrarla sempre, qualunque sia il suo valore.

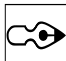
Per specificare un valore predefinito per una proprietà:

1. Aggiungere un segno di uguale (=) dopo il nome della proprietà.
2. Dopo il segno di uguale, aggiungere una coppia di parentesi graffe ({}).
3. All'interno delle graffe, scrivere la parola chiave **default** seguita da un altro segno di uguale.
4. Specificare il nuovo valore predefinito.

Ad esempio:

```
__property Alignment = {default=taCenter};
```

È possibile specificare un valore predefinito anche quando si dichiara una proprietà. Infatti, una ragione per ridichiarare una proprietà è quella di indicare un valore predefinito diverso.



La specifica del valore predefinito di una proprietà non assegna automaticamente quel valore alla proprietà in fase di creazione di un oggetto. Occorre assicurarsi che il costruttore del componente assegni il valore necessario. Una proprietà, il cui valore non è impostato dal costruttore, assume come valore zero, cioè, qualunque valore assunto dalla proprietà quando l'area di memorizzazione viene impostata a 0. In altre parole, i valori numerici hanno come impostazione predefinita 0, i valori booleani sono **false**, i puntatori sono **NULL**, e così via. In caso di dubbio, assegnare un valore nel costruttore del metodo.

Il codice seguente mostra la dichiarazione di un componente che specifica un valore predefinito per la proprietà *Align* e l'implementazione da parte del costruttore del componente che imposta il valore predefinito. In questo caso, il nuovo componente è un caso speciale di un componente panel standard che sarà utilizzato per le barre di stato di una finestra, in modo che l'allineamento predefinito sia il fondo dell'oggetto proprietario.

```
class PACKAGE TMyStatusBar : public TPanel
{
public:
  virtual __fastcall TMyStatusBar(TComponent* AOwner);
__published:
  __property Align = {default=alBottom};
};
```

Il costruttore del componente *TMyStatusBar* nel file .CPP è il seguente:

```
__fastcall TMyStatusBar::TMyStatusBar (TComponent* AOwner)
  : TPanel(AOwner)
{
   Align = alBottom;
}
```

Determinazione di che cosa registrare

È possibile controllare se C++Builder registra tutte le proprietà dei componenti. Come impostazione predefinita, tutte le proprietà presenti nella parte published della dichiarazione di classe vengono registrate. Si può scegliere di non registrare una determinata proprietà, oppure si può designare una funzione che determini se registrare o meno una proprietà.

Per controllare se C++Builder debba registrare o meno una proprietà:

1 Aggiungere un segno di uguale (=) dopo il nome della proprietà.
2 Dopo il segno di uguale, aggiungere una coppia di parentesi graffe({}).
3 All'interno delle graffe, scrivere la parola chiave **stored** seguita da **true**, **false** o dal nome di un metodo booleano.

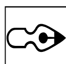
Il codice seguente mostra un componente che dichiara tre nuove proprietà. Una viene registrata sempre, una non viene mai registrata, mentre la terza viene registrata a seconda del valore di una funzione booleana:

```
class PACKAGE TSampleComponent : public TComponent
{
protected:
  bool __fastcall StoreIt();
public:
   ⋮
__published:
  __property int Important = {stored=true};         // always stored
  __property int Unimportant = {stored=false};      // never stored
  __property int Sometimes = {stored=StoreIt};      // storage depends on function value
};
```

Inizializzazione dopo il caricamento

Un componente, dopo aver letto tutti i valori delle sue proprietà dalla sua descrizione registrata, richiama un metodo virtuale chiamato *Loaded*, che esegue tutte le inizializzazioni richieste. La chiamata di *Loaded* avviene prima che compaiano la scheda e i suoi controlli, così che non c'è da preoccuparsi del fatto che l'inizializzazione possa provocare sfarfallio sullo schermo.

Per inizializzare un componente dopo che ha caricato i valori delle sue proprietà, occorre ridefinire il metodo *Loaded*.



La prima cosa da fare in un metodo *Loaded* è chiamare il metodo ereditato *Loaded*. Ciò assicura che tutte le proprietà ereditate verranno inizializzate correttamente prima che sia inizializzato il componente.

Registrazione e caricamento di proprietà non pubbliche

Per impostazione predefinita, sole le proprietà pubbliche vengono caricate e salvate con un componente. Tuttavia, è possibile caricare e salvare anche proprietà non pubbliche. Questa possibilità consente di avere proprietà persistenti che non appaiono nell'Object Inspector. Ciò consente anche ai componenti di memorizzare e caricare valori di proprietà che C++Builder non è in grado di leggere o di scrivere perché il valore della proprietà è troppo complesso. Per esempio, l'oggetto *TStrings* non può appoggiarsi al comportamento automatico di C++Builder per memorizzare e caricare le stringhe che esso rappresenta e deve utilizzare il meccanismo spiegato di seguito.

È possibile salvare le proprietà non pubbliche aggiungendo del codice che spiega a C++Builder come caricare e salvare il valore delle proprietà.

Per scrivere il codice necessario a caricare e salvare le proprietà, seguire le istruzioni seguenti:

1 Creare dei metodi per memorizzare e caricare il valore della proprietà.

2 Ridefinire il metodo *DefineProperties*, passando quei metodi un oggetto filer.

Creazione di metodi per memorizzare e caricare i valori delle proprietà

Per memorizzare e caricare le proprietà non pubbliche, per prima cosa si deve creare un metodo per memorizzare il valore della proprietà e un altro metodo per caricarlo. Esistono due possibilità:

- Creare un metodo di tipo *TWriterProc* per memorizzare il valore della proprietà e un metodo di tipo *TReaderProc* per caricare il valore della proprietà. Questo approccio consente di sfruttare le capacità intrinseche di C++Builder di salvataggio e caricamento di tipi semplici. Se il valore della proprietà è costituito da tipi che C++Builder è in grado di salvare e caricare, utilizzare questo metodo.
- Creare due metodi di tipo *TStreamProc*, uno per memorizzare e uno per caricare il valore della proprietà. *TStreamProc* richiede come un argomento uno stream, ed è

possibile utilizzare i metodi dello stream per scrivere e leggere i valori della proprietà.

Per esempio, si immagini una proprietà che rappresenta un componente creato in fase di esecuzione. C++Builder sa come scrivere questo valore, ma non è in grado di farlo automaticamente perché il componente non è stato creato da Form Designer. Poiché il sistema di stream è già in grado di caricare e salvare componenti, è possibile utilizzare il primo metodo.I metodi seguenti caricano e memorizzano il componente creato dinamicamente che rappresenta il valore di una proprietà di nome *MyCompProperty*:

```
void __fastcall TSampleComponent::LoadCompProperty(TReader *Reader)
{
  if (Reader->ReadBoolean())
    MyCompProperty = Reader->ReadComponent(NULL);
}
void __fastcall TSampleComponent::StoreCompProperty(TWriter *Writer)
{
  if (MyCompProperty)
  {
    Writer->WriteBoolean(true);
    Writer->WriteComponent(MyCompProperty);
  }
  else
    Writer->WriteBoolean(false);
}
```

Ridefinizione del metodo DefineProperties

Una volta creati i metodi per memorizzare e caricare il valore della proprietà, è possibile ridefinire il metodo *DefineProperties* del componente. C++Builder chiama questo metodo quando carica o registra il componente. Nel metodo *DefineProperties*, si deve chiamare il metodo *DefineProperty* o il metodo *DefineBinaryProperty* del filer corrente, passandogli il metodo da utilizzare per caricare o per salvare il valore della proprietà. Se i metodi di registrazione e caricamento sono di tipo *TWriterProc* e di tipo *TReaderProc*, si chiamerà il metodo *DefineProperty* del filer. Se i metodi creati sono di tipo *TStreamProc*, si chiamerà invece *DefineBinaryProperty*.

Indipendentemente dal tipo metodo utilizzato per definire la proprietà, lo si passa ai metodi che memorizzano e caricano il valore della proprietà, assieme ad un valore di tipo boolean che indica se il valore della proprietà deve essere scritto. Se il valore può essere ereditato o ha un valore predefinito, non sarà necessario scriverlo.

Per esempio, dato il metodo *LoadCompProperty* di tipo *TReaderProc* e il metodo *StoreCompProperty* di tipo *TWriterProc*, si potrebbe ridefinire *DefineProperties* nel seguente modo:

```
void __fastcall TSampleComponent::DefineProperties(TFiler *Filer)
{
  // before we do anything, let the base class define its properties.
  // Note that this example assumes that TSampleComponent derives directly from TComponent
  TComponent::DefineProperties(Filer);
  bool WriteValue;
  if (Filer->Ancestor) // check for inherited value
```

```
  {
    if ((TSampleComponent *)Filer->Ancestor)->MyCompProperty == NULL)
      WriteValue = (MyCompProperty != NULL);
    else if ((MyCompProperty == NULL) ||
             (((TSampleComponent *)Filer->Ancestor)->MyCompProperty->Name !=
                                                   MyCompProperty->Name))
      WriteValue = true;
    else WriteValue = false;
  }
  else // no inherited value, write property if not null
    WriteValue = (MyCompProperty != NULL);
  Filer->DefineProperty("MyCompProperty ",LoadCompProperty,StoreCompProperty, WriteValue);
end;
```

Capitolo

48

Creazione di eventi

Un evento è un collegamento tra un avvenimento nel sistema (ad esempio un'azione dell'utente o una modifica del fuoco) e una parte di codice che risponde a quel determinato avvenimento. Il codice che risponde è un gestore di evento, e quasi sempre è scritto da chi sviluppa l'applicazione. Gli eventi consentono agli sviluppatori di personalizzare il comportamento dei componenti senza modificare le relative classi. Ciò è noto come *delegation*.

Gli eventi per le azioni più comuni dell'utente (come le azioni del mouse) sono intrinseci a tutti i componenti standard, ma si possono definire anche nuovi eventi. Per creare eventi in un componente, è necessario capire quanto segue:

- Cos'è un evento
- Implementazione degli eventi standard
- Definizione di eventi personalizzati

Gli eventi sono implementati come proprietà e pertanto, prima di provare a creare o modificare gli eventi di un componente, si dovrebbe già avere familiarità con gli argomenti trattati nel Capitolo 47, "Creazione di proprietà",.

Cos'è un evento

Un evento è un meccanismo che collega un avvenimento al codice. Più esattamente, un evento è un closure che punta a un metodo in una determinata istanza della classe.

Dal punto di vista dello sviluppatore di applicazioni, un evento è soltanto un nome che si riferisce a un avvenimento del sistema, come *OnClick*, a cui è possibile collegare una determinata sequenza di istruzioni. Per esempio, un pulsante di nome *Button1*, ha un metodo *OnClick*. Per impostazione predefinita, C++Builder genera un gestore di evento denominato *Button1Click* nella scheda che contiene il pulsante e lo assegna a *OnClick*. Quando nel pulsante si verifica un evento clic, il pulsante richiama il metodo assegnato a *OnClick*, in questo caso *Button1Click*.

Per scrivere un evento occorre capire quanto segue:

- Gli eventi sono closure.
- Gli eventi sono proprietà.
- I tipi degli eventi sono tipi closure.
- Il tipo restituito dai gestori di evento è void
- I gestori di evento sono facoltativi.

Gli eventi sono closure

C++Builder utilizza i closure per implementare gli eventi. Un closure è un tipo speciale di puntatore che punta ad un certo metodo in una certa istanza di una classe. Chi scrive componenti può trattare il closure come un segnaposto. Il programma rileva il verificarsi di un evento, e pertanto chiama (se esiste) il metodo specificato dall'utente per quell'evento.

I closure mantengono un puntatore nascosto a un'istanza della classe. Quando l'utente assegna un gestore all'evento di un componente, l'assegnazione non è soltanto a un metodo con un determinato nome, ma piuttosto a un metodo specifico di una specifica istanza di una classe. Questa istanza è di solito la scheda contenente il componente, ma non necessariamente.

Tutti i controlli, per esempio, ereditano un metodo virtuale di nome *Click* per gestire gli eventi clic:

```
virtual void __fastcall Click(void);
```

L'implementazione di *Click* chiama il gestore di eventi clic dell'utente, se ne esiste uno. Se l'utente ha assegnato un gestore a un evento *OnClick* del controllo, facendo clic sul controllo viene chiamato quel metodo. Se non è stato assegnato nessun gestore, non accade nulla.

Gli eventi sono proprietà

I componenti usano le proprietà per implementare i propri eventi. A differenza della maggior parte delle proprietà, gli eventi non usano metodi per implementare le proprie parti read e write. Invece, le proprietà dell'evento usano un membro dati privato della classe il cui tipo è uguale a quello della proprietà.

Per convenzione, il nome del membro dati è lo stesso nome della proprietà, ma preceduto dalla lettera F. Ad esempio, il closure *OnClick* è memorizzato in un membro dati di nome *FOnClick* di tipo *TNotifyEvent* e la dichiarazione della proprietà evento *OnClick* è simile alla seguente:

```
class PACKAGE TControl : public TComponent
{
private:
    TNotifyEvent FOnClick;
  ⋮
protected:
    __property TNotifyEvent OnClick = {read=FOnClick, write=FOnClick};
  ⋮
```

```
};
```

Per saperne di più su *TNotifyEvent* e sugli altri tipi di evento, vedere la prossima sezione "I tipi degli eventi sono tipi closure".

valore dell'evento anche durante l'esecuzione. Il vantaggio principale della gestione degli eventi come proprietà, comunque, è che l'utente del componente può assegnare all'evento un gestore durante lo sviluppo usando Object Inspector.

I tipi degli eventi sono tipi closure

Poiché un evento è un puntatore a un gestore di eventi, il tipo della proprietà dell'evento deve essere un closure. Analogamente, il codice usato per gestire l'evento deve essere il metodo di una classe, di tipo appropriato.

Per essere compatibile con un evento di un certo tipo, il metodo di gestione evento deve avere lo stesso numero e tipo di parametri, nello stesso ordine e passati nello stesso modo.

C++Builder definisce i closure per tutti i suoi eventi standard. Quando il programmatore crea i propri eventi, può utilizzare un closure esistente, se è appropriato, o definirne uno proprio.

Il tipo restituito dai gestori di evento è void

Il tipo restituito dai gestori di evento può essere solo di tipo void. Anche se i gestori possono restituire solo tipi void, è sempre possibile ottenere dal codice utente delle informazioni di ritorno passando gli argomenti per riferimento. Se si adotta questo metodo, occorre accertarsi di assegnare all'argomento un valore consentito prima di effettuare la chiamata, in modo che non sia necessario che il codice dell'utente modifichi il valore.

Un esempio del passaggio degli argomenti per riferimento a un gestore di evento è l'evento legato alla pressione dei tasti, di tipo *TKeyPressEvent*. *TKeyPressEvent* definisce due argomenti, il primo indicante l'oggetto che ha generato l'evento, e l'altro indicante il tasto premuto:

```
typedef void __fastcall (__closure *TKeyPressEvent)(TObject *Sender, Char &Key);
```

Normalmente, il parametro *Key* contiene il carattere premuto dall'utente. In determinate circostanze, tuttavia, l'utente del componente potrebbe decidere di cambiare il carattere. Un esempio potrebbe essere quello di forzare tutti i caratteri in maiuscolo in un controllo per l'editing. In questo caso l'utente potrebbe definire il seguente gestore per la pressione dei tasti:

```
void __fastcall TForm1::Edit1KeyPress(TObject *Sender, Char &Key)
{
  Key = UpCase(Key);
}
```

È anche possibile utilizzare argomenti passati per riferimento in modo da consentire all'utente di ridefinire la gestione predefinita.

I gestori di evento sono facoltativi

Nella creazione di eventi, occorre ricordare che gli sviluppatori che utilizzano i componenti sviluppati da altri potrebbero non collegare ad essi un gestore di eventi. Ciò significa che il componente non dovrebbe bloccarsi o generare errori solo perché a un particolare evento non è collegato un gestore. (I meccanismi di chiamata di un gestore evento e le modalità operative con eventi che non hanno un gestore di eventi collegato sono spiegati nella sezione “Chiamata dell’evento” a pagina 48-9.)

Gli eventi si verificano quasi ininterrottamente in un’applicazione GUI. Il semplice spostamento del puntatore del mouse su un componente visuale, provova la trasmissione di numerosi messaggi di movimento mouse che il componente traduce in eventi *OnMouseMove*. Nella maggior parte dei casi, gli sviluppatori non vogliono gestire gli eventi di movimento del mouse, e ciò non dovrebbe comportare problemi. Perciò i componenti creati non dovrebbero prevedere un gestore per il trattamento dei relativi eventi.

Inoltre, gli sviluppatori di applicazioni possono scrivere una qualsiasi sequenza di istruzioni in un gestore di evento. I componenti nella VCL hanno eventi scritti in modo da minimizzare i rischi che un gestore di evento possa generare errori. Naturalmente, è impossibile garantirsi contro errori di logica nel codice dell’applicazione, ma si può fare in modo di controllare che le strutture dati vengano inizializzate prima di effettuare chiamate agli eventi, in modo che chi sviluppa non possa accedere a dati non validi.

Implementazione degli eventi standard

I controlli forniti con C++Builder ereditano gli eventi per gli avvenimenti più frequenti. Questi sono chiamati eventi standard. Benché tutti questi eventi siano inclusi nei controlli, spesso sono **protected**, vale a dire che gli sviluppatori non possono collegare ad essi alcun gestore di eventi. Quando si crea un controllo, è possibile scegliere se rendere visibili gli eventi agli utenti del controllo.

Vi sono tre aspetti da considerare quando si incorporano gli eventi standard all’interno del controllo:

- Identificazione degli eventi standard
- Visibilità degli eventi
- Modifica della gestione standard degli eventi

Identificazione degli eventi standard

Esistono due categorie di eventi standard: quelli definiti per tutti i controlli e quelli definiti solo per i controlli standard per finestra.

Eventi standard di tutti i controlli

Gli eventi più comuni sono definiti nella classe *TControl*. Tutti i controlli, siano essi per finestra, grafici o personalizzati, ereditano questi eventi. I seguenti eventi sono disponibili in tutti i controlli:

OnClick	*OnDragDrop*	*OnEndDrag*	*OnMouseMove*
OnDblClick	*OnDragOver*	*OnMouseDown*	*OnMouseUp*

Gli eventi standard hanno metodi virtual protected corrispondenti dichiarati in *TControl*, con nomi che corrispondono ai nomi dell'evento. Per esempio, gli eventi *OnClick* chiamano un metodo di nome *Click*, mentre gli eventi *OnEndDrag* chiamano un metodo di nome *DoEndDrag*.

Gli eventi standard dei controlli standard

Oltre agli eventi comuni a tutti i controlli, i controlli con finestra standard (quelli che derivano da *TWinControl* nella VCL e *TWidgetControl* nella CLX) possiedono i seguenti eventi:

OnEnter	*OnKeyDown*	*OnKeyPress*
OnKeyUp	*OnExit*	

Analogamente agli eventi in *TControl*, anche gli eventi dei controlli con finestra hanno metodi corrispondenti. Gli eventi standard di tastiera elencati in precedenza rispondono a tutte le normali pressioni dei tasti.

Comunque, per rispondere a speciali combinazioni di tasti (come quelle con il tasto Alt), è necessario rispondere al messaggio WM_GETDLGCODE o CM_WANTSPECIALKEYS da Windows. Per informazioni sulla scrittura dei gestori di messaggi vedere il Capitolo 51, "Gestione dei messaggi e delle notifiche di sistema".

Visibilità degli eventi

Le dichiarazioni degli eventi standard in *TControl* e *TWinControl* (*TWidgetControl* nella CLX) sono protette, come lo sono i metodi ad essi corrispondenti. Se si sta ereditando da una di queste classi astratte e si vogliono rendere accessibili i relativi eventi in esecuzione o in progettazione, occorre ridichiarare gli eventi come public o published.

La ridichiarazione di una proprietà senza specificarne l'implementazione mantiene invariati i metodi d'implementazione, ma modifica il livello di protezione. È quindi possibile prendere un evento definito in *TControl* ma che non è stato reso visibile, e renderlo tale dichiarandolo public o published.

Ad esempio, per creare un componente che rende visibile l'evento *OnClick* in fase di progetto, si potrebbe aggiungere quanto segue alla dichiarazione della classe del componente.

```
class PACKAGE TMyControl : public TCustomControl
```

```
{
  :
__published:
    __property OnClick;              // Makes OnClick available in the Object Inspector
};
```

Modifica della gestione standard degli eventi

IVolendo modificare il modo in cui il componente risponde a determinati tipi di evento, si potrebbe essere tentati di scrivere una certa sequenza di istruzioni e assegnarla all'evento. Come sviluppatori di applicazioni, è esattamente quello si farebbe. Ma quando si crea un componente, occorre mantenere l'evento disponibile per gli sviluppatori che utilizzeranno il componente.

Questo è il motivo dei metodi di implementazione protetti, associati a ognuno degli eventi standard. Ridefinendo il metodo di implementazione, è possibile cambiare la gestione interna dell'evento; e chiamando il metodo ereditato, è possibile conservare la gestione standard, incluso l'evento per il codice dello sviluppatore di applicazioni.

L'ordine in cui si chiamano i metodi è importante. Come regola, occorre chiamare prima il metodo ereditato, consentendo che l'esecuzione del gestore di evento previsto dallo sviluppatore dell'applicazione avvenga prima della propria personalizzazione (e, in alcuni casi, impedendo che avvenga). Possono comunque verificarsi casi in cui si vuole che il proprio codice venga eseguito prima di chiamare il metodo ereditato. Per esempio, se il codice ereditato è in qualche modo dipendente dallo stato del componente e il proprio codice modifica lo stato, occorrerebbe effettuare le modifiche e poi consentire al codice dell'utente di rispondere ad esse.

Si supponga che si stia scrivendo un componente e si voglia modificare il modo con cui risponde al clic del mouse. Invece di assegnare un gestore all'evento *OnClick* come farebbe uno sviluppatore di applicazioni, si può sostituire il metodo *Click* protetto:

```
void __fastcall TMyControl::Click()
{
  TWinControl::Click();        // perform standard handling, including calling handler
  // your customizations go here
}
```

Definizione di eventi personalizzati

La definizione di eventi completamente nuovi è una cosa abbastanza insolita. Ci sono comunque situazioni in cui, quando un componente presenta un comportamento completamente diverso da quello di qualsiasi altro componente, è necessario definire un evento specifico.

Nella definizione di un evento vanno considerati i seguenti aspetti:

- Attivazione dell'evento
- Definizione del tipo di gestore

- Dichiarazione dell'evento
- Chiamata dell'evento

Attivazione dell'evento

Occorre sapere che cosa provoca l'attivazione di un evento. Per alcuni eventi, la risposta è ovvia. Per esempio, un evento mouse-down si verifica quando l'utente preme il pulsante sinistro del mouse e Windows invia un messaggio *WM_LBUTTONDOWN* all'applicazione. Alla ricezione di questo messaggio, un componente chiama il proprio metodo *MouseDown* che, a sua volta, chiama il codice che l'utente ha associato all'evento *OnMouseDown*.

Ma ci sono eventi chiaramente dipendenti da specifici avvenimenti esterni. Per esempio, un evento *OnChange* della barra di scorrimento è provocato da avvenimenti di vario genere, inclusi la pressione di tasti, il clic del mouse o modifiche in altri controlli. Nella definizione degli eventi bisogna garantire che tutti gli avvenimenti richiamino in modo appropriato i rispettivi eventi.

Per le applicazioni CLX, vedere "Risposta alle notifiche di sistema utilizzando la CLX" a pagina 51-11.

Due tipi di eventi

Vi sono due tipi di avvenimenti per cui sarebbe necessario prevedere eventi: interazioni dell'utente e cambiamenti di stato. Gli eventi di interazione dell'utente sono quasi sempre attivati da un messaggio di Windows, che indica che l'utente ha fatto qualcosa cui il componente potrebbe dover rispondere. Gli eventi di cambiamenti di stato possono anch'essi essere correlati a messaggi di Windows (cambiamento o attivazione del fuoco, per esempio), ma possono verificarsi anche per modifiche alle proprietà o in altre situazioni.

Occorre avere il controllo totale di ciò che può attivare un evento definito. Bisogna definire con precisione gli eventi, in modo che gli sviluppatori siano in grado di capirli e di utilizzarli

Definizione del tipo di gestore

Una volta determinato quando si verifica l'evento, occorre definire come deve essere gestito l'evento. Ciò significa stabilire il tipo di gestore di evento. Nella maggior parte dei casi i gestori degli eventi definiti da chi programma sono semplici notifiche oppure tipi che dipendono dell'evento. È anche possibile fare in modo che il gestore restituisca delle informazioni.

Notifica semplice

Un evento di notifica informa che l'evento è accaduto, ma non dice quando o dove è accaduto. Le notifiche utilizzano il tipo *TNotifyEvent*, che prevede un solo parametro, e cioè chi ha scatenato l'evento. Tutto ciò che gestore di una notifica "conosce" a proposito dell'evento è il tipo di evento e per quale componente l'evento si è verificato. Per esempio, gli eventi clic sono notifiche. Quando si scrive un gestore per

un evento clic tutto ciò che si sa è che si è verificato un clic e su quale componente è stato fatto clic.

La notifica è un processo monodirezionale. Non esiste alcun meccanismo per fornire informazioni di ritorno o per prevenire un'ulteriore gestione di una notifica.

Gestori che dipendono dall'evento

In alcuni casi non è sufficiente conoscere quale evento si è verificato e su quale componente si è verificato. Per esempio, se l'evento è legato alla pressione di un tasto, è probabile che il gestore di eventi voglia sapere qual è il tasto premuto dall'utente. In questi casi occorrono dei tipi di gestore che includano parametri per informazioni aggiuntive.

Se l'evento è stato generato in risposta a un messaggio, è probabile che i parametri passati al gestore dell'evento siano presi direttamente dai parametri del messaggio.

Restituzione di informazioni da parte del gestore

Poiché tutti i gestori di evento restituiscono solo void, il solo modo che ha un gestore per restituire informazioni è tramite parametri passati per riferimento. I componenti potranno utilizzare queste informazioni per determinare come o se trattare l'evento, una volta terminata l'esecuzione del gestore definito dall'utente.

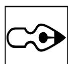
Ad esempio, tutti gli eventi relativi ai tasti (*OnKeyDown*, *OnKeyUp* e *OnKeyPress*) passano per riferimento il valore del tasto premuto in un parametro var di nome *Key*. Il gestore di evento può modificare *Key* in modo che l'applicazione riconosca come tasto implicato nell'evento un tasto diverso da quello utilizzato. Questo è, ad esempio, il modo per forzare la scrittura di caratteri in maiuscolo.

Dichiarazione dell'evento

Una volta determinato il tipo di gestore evento, si può iniziare a dichiarare il closure e le proprietà dell'evento. È consigliabile utilizzare dei nomi significativi e descrittivi in modo che l'utente possa capire a cosa serve l'evento. È anche importante la coerenza e quindi bisogna cercare di dare nomi analoghi a quelli delle proprietà simili di altri componenti.

I nomi degli eventi cominciano con "On"

The I nomi di molti eventi in C++Builder cominciano con "On". Questa è soltanto una convenzione; il compilatore non impone di attenervisi. L'Object Inspectordetermina che una proprietà è un evento esaminando il tipo della proprietà: tutte le proprietà di closure vengono considerate eventi e appaiono nella pagina Events.
Gli sviluppatori si aspettano di trovare gli eventi nell'elenco alfabetico dei nomi che cominciano con "On". Usando altri tipi di nomi, si rischierebbe di confonderli.



L'eccezione principale a questa regola è che molti eventi che si verificano prima di e dopo un certo accadimento iniziano con "Before" e "After".

Chiamata dell'evento

Le chiamate a un evento dovrebbero essere centralizzate. Cioè, si dovrebbe creare nel componente un metodo virtual che chiami il gestore di eventi dell'applicazione (se ne è stato assegnato uno) e che fornisca la gestione predefinita.

Collocare tutte le chiamate dell'evento in un punto garantisce che chiunque derivi un nuovo componente possa personalizzare la gestione dell'evento sovrapponendosi a un unico metodo, anziché ricercare in tutto il programma tutte le posizioni da cui viene chiamato l'evento.

Vi sono due altre considerazioni da fare quando si chiama l'evento:

- I gestori di evento vuoti devono essere validi.
- Gli utenti possono ridefinire la gestione predefinita.

I gestori di evento vuoti devono essere validi

Non si dovrebbe mai creare una situazione in cui un gestore evento vuoto possa causare un errore, nemmeno se il funzionamento corretto del componente dipende da una particolare risposta derivante dal codice di gestione evento dell'applicazione.

Un gestore di evento vuoto dovrebbe produrre lo stesso risultato che si avrebbe se non ci fosse alcun gestore evento. Perciò la sequenza di istruzioni per chiamare il gestore di evento di un'applicazione dovrebbe apparire così:

```
if (OnClick)
  OnClick(this);
// perform default handling }
```

Non si dovrebbe mai avere un codice simile al seguente:

```
if (OnClick)
  OnClick(this);
else
  // perform default handling
```

Gli utenti possono ridefinire la gestione predefinita

Per alcuni tipi di eventi lo sviluppatore potrebbe voler sostituire la gestione predefinita, o addirittura sopprimere tutte le risposte. Per fare ciò, è necessario passare per riferimento un argomento al gestore di evento e, al ritorno dal gestore, controllare un determinato valore.

Questo è in linea con la regola che un gestore di evento vuoto dovrebbe produrre lo stesso effetto che si avrebbe se non ci fosse alcun gestore di evento. Poiché un gestore vuoto non modificherà i valori degli argomenti passati per riferimento, la gestione predefinita verrà attivata sempre dopo avere chiamato il gestore vuoto.

Ad esempio, quando si gestiscono gli eventi legati alla pressione dei tasti, l'utente può sopprimere la gestione predefinita della pressione del tasto effettuata dal componente impostando il parametro *Key* a un carattere NULL.La logica per supportare ciò è simile all'esempio seguente:

```
if (OnKeyPress)
```

```
  OnKeyPress(this, &Key);
if (Key != NULL)
  //perform default handling
```

Il codice vero e proprio sarà leggermente diverso da questo in quanto deve trattare i messaggi di Windows, ma la logica è la stessa. Per impostazione predefinita, il componente chiama qualsiasi gestore assegnato dall'utente e poi esegue la sua gestione standard. Se il gestore definito dall'utente imposta *Key* a un carattere NULL, il componente non esegue la gestione predefinita.

Capitolo

49

Creazione di metodi

I metodi dei componenti non sono molto differenti dai metodi delle altre classi. Ciò significa che essi sono funzioni membro incorporate nella struttura della classe di un componente. Benché non vi siano sostanziali restrizioni su ciò che è possibile fare con i metodi di un componente, C++Builder utilizza alcuni standard che andrebbero seguiti. Questi sono:

- Eliminazione delle dipendenze
- Assegnazione del nome ai metodi
- Protezione dei metodi
- Utilizzo dei metodi virtual
- Dichiarazione dei metodi

In generale, i componenti non dovrebbero contenere molti metodo e si dovrebbe ridurre al minimo il numero di metodi che un'applicazione deve chiamare. Le caratteristiche che si sarebbe tentati di implementare come metodi spesso vengono incapsulati meglio nelle proprietà. Le proprietà mettono a disposizione un'interfaccia adatta all'ambiente di C++Builder e sono accessibili in fase di progetto.

Eliminazione delle dipendenze

Quando si scrive il codice di un componente è auspicabile ridurre al minimo le precondizioni imposte allo sviluppatore. Per quanto possibile, gli sviluppatori dovrebbero poter utilizzare il componente in qualsiasi contesto e in qualsiasi momento. Ci saranno certamente circostanze in cui non sarà possibile soddisfare questa regola, ma l'obiettivo di chi sviluppa componenti è quello di attenervisi il più possibile.

Questo elenco dà un'idea dei tipi di dipendenze da evitare:

- Metodi che l'utente *deve* richiamare per utilizzare il componente
- Metodi che vanno eseguiti rispettando una particolare sequenza

- Metodi che mettono il componente in uno stato o in una condizione in cui determinati eventi o metodi possono risultare non validi

Il migliore modo per gestire queste situazioni è quello di fornire un metodo per uscirne. Per esempio, se chiamando un metodo si mette il componente in un stato che risulterebbe non valido se si chiamasse un altro metodo, occorre scrivere quel secondo metodo in modo tale che corregga lo stato del componente nel caso venga chiamato da un'applicazione quando lo stato del componente non è corretto, prima di eseguire il codice principale. Come minimo, si dovrebbe sollevare un'eccezione nei casi in cui un utente chiama un metodo che non è valido.

In altri termini, se si crea una situazione in cui delle parti di codice dipendono l'una dall'altra, il *programmatore* dovrebbe farsene carico per evitare che un utilizzo scorretto del codice non provochi problemi. Per esempio, nel caso in cui l'utente non rispetti le dipendenze imposte, un messaggio di segnalazione è preferibile a un errore di sistema.

Assegnazione del nome ai metodi

C++Builder non impone restrizioni sui nomi assegnati ai metodi o ai rispettivi parametri. Vi sono comunque alcune convenzioni che semplificano l'uso dei metodi a chi sviluppa applicazioni. È bene ricordare che è la stessa architettura a componenti che fa sì che i componenti sviluppati possano essere utilizzati da vari tipi di utenti.

Chi è abituato a scrivere codice per uso personale o per un piccolo gruppo di programmatori potrebbe essere tentato di trascurare questo aspetto della programmazione. È comunque consigliabile cercare di rendere i nomi dei metodi più chiari possibile, in modo che anche le persone che non hanno familiarità con il codice (o addirittura con la programmazione in assoluto), siano poste in condizione di usare i componenti.

Ecco alcuni suggerimenti per rendere più chiari i nomi dei metodi:

- Usare nomi descrittivi. Utilizzare verbi significativi.

  Un nome come *PasteFromClipboard* offre molte più informazioni di un semplice *Paste* o *PFC*.

- I nomi delle funzioni dovrebbero riflettere la natura del risultato restituito.

  Può anche sembrare ovvio che una funzione chiamata *X* restituisca la posizione orizzontale sullo schermo di un elemento, ma un nome come *GetHorizontalPosition* è più facilmente comprensibile.

- Se il tipo restituito da una funzione è void, il nome della funzione dovrebbe essere attivo.

  Nei nomi delle funzioni si consiglia l'utilizzo di verbi attivi. Ad esempio, *ReadFileNames* è molto più comprensibile di *DoFiles*.

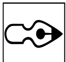
Una considerazione finale: verificare che il metodo debba essere necessariamente un metodo. Una regola empirica è che il nome del metodo contenga un verbo. Se si

scopre che si stanno creando molti metodi il cui nome non contiene un verbo, è bene prendere in esame il fatto se trasformare tutti questi metodi in proprietà.

Protezione dei metodi

Tutte le parti delle classi, compresi i membri dei dati, i metodi e le proprietà hanno un livello di protezione o di "visibilità", come spiegato nella sezione "Controllo dell'accesso" a pagina 46-4. Scegliere il tipo di visibilità appropriata per un determinato metodo è semplice.

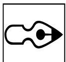
La maggior parte dei metodi che lo sviluppatore scrive nei componenti sono **public** o **protected**. Raramente si presenta la necessità di rendere **private** un metodo, a meno che non sia veramente specifico per quel tipo di componente, al punto che anche i componenti derivati non possono avere accesso ad esso.



Di solito, non c'è alcun motivo per dichiarare un metodo (ad eccezione di un gestore di evento) come **__published**. Così facendo, dal punto di vista dell'utente, sarebbe come se il metodo fosse stato dichiarato **public**.

Metodi che devono essere pubblici

Tutti i metodi che gli sviluppatore di applicazioni hanno bisogno di richiamare devono essere dichiarati public. Si deve tenere presente che la maggior parte delle chiamate ai metodi avviene nei gestori di eventi, per cui i metodi dovrebbe evitare di creare vincoli alle risorse del sistema o di mettere il sistema operativo in condizione di non poter rispondere all'utente.



I costruttori e i distruttori devono essere sempre **public**.

Metodi che devono essere protetti

Ogni metodo implementato per un componente deve essere dichiarato **protected** in modo tale che le applicazioni non lo possano chiamare al momento sbagliato. Se si hanno dei metodi che non devono essere chiamati dal codice dell'applicazione, ma che sono chiamati da classi derivate, bisogna dichiararli **protected**.

Ad esempio, si supponga di avere un metodo che prevede la preimpostazione di alcuni dati. Se si rende **public** tale metodo, esiste la possibilità che applicazioni lo chiamino prima che i dati siano stati impostati. D'altra parte, rendendolo **protected** si è sicuri che le applicazioni non lo possano chiamare direttamente. Si possono, quindi, impostare altri metodi **public**, in modo da assicurare che l'impostazioni dei dati avvenga prima della chiamata al metodo **protected**.

I metodi che implementano le proprietà devono essere dichiarati metodi virtual **protected**. I metodi così dichiarati permettono agli sviluppatori di applicazioni di ridefinire l'implementazione della proprietà, aumentandone la funzionalità o sostituendola completamente. Tali proprietà sono completamente polimorfe. Mantenendo i metodi di accesso **protected**, si ha la certezza che gli sviluppatori non li possano chiamare accidentalmente, modificando inavvertitamente una proprietà.

Utilizzo dei metodi virtual

I metodi vengono resi **virtual** quando si vuole che tipi differenti siano in grado di eseguire sequenze di istruzioni diverse in risposta alla stessa chiamata di metodo.

Se si creano componenti destinati a essere utilizzati direttamente da chi sviluppa applicazioni, è possibile rendere non virtual tutti i metodi. D'altra parte, se si creano componenti astratti da cui saranno derivati altri componenti, è bene rendere **virtual** i metodi aggiunti. In questo modo, i componenti derivati possono ridefinire i metodi **virtual** ereditati.

Dichiarazione dei metodi

La dichiarazione di un metodo in un componente è analoga alla dichiarazione di un qualsiasi metodo di una classe.

Per dichiarare un nuovo metodo in un componente, si deve fare quanto segue:

- Aggiungere la dichiarazione del metodo alla dichiarazione della classe del componente nel file header del componente.
- Scrivere nel file .CPP della unit il codice che implementa il metodo.

Il codice seguente mostra un componente che definisce due nuovi metodi, un metodo **protected** e un metodo virtual **public**. Quella che segue è la definizione nel file .H dell'interfaccia:

```
class PACKAGE TSampleComponent : public TControl
{
protected:
    void __fastcall MakeBigger();
public:
    virtual int __fastcall CalculateArea();
  ⋮
};
```

Quello che segue è il codice, incluso nel file .CPP, che implementa i metodi:

```
void __fastcall TSampleComponent::MakeBigger()
{
  Height = Height + 5;
  Width = Width + 5;
}

int __fastcall TSampleComponent::CalculateArea()
{
  return Width * Height;
}
```

Capitolo

50

Uso della grafica nei componenti

Windows dispone di una potente interfaccia GDI (graphics device interface) per disegnare grafici indipendenti dal dispositivo. La GDI richiede al programmatore ulteriori requisiti, come ad esempio la gestione delle risorse grafiche. C++Builder si fa carico di tutte le fastidiose incombenze necessarie alla GDI, permettendo di concentrarsi sul lavoro produttivo invece di andare alla ricerca di handle persi o di risorse non rilasciate.

Le funzioni GDI, come qualsiasi parte delle API di Windows, possono essere chiamate direttamente dall'applicazione C++Builder. Molto probabilmente, però, si noterà che è più rapido e semplice usare l'incapsulamento delle funzioni grafiche di C++Builder.

Le funzioni GDI sono specifiche per Windows e non sono utilizzabili in ambiente CLX o per applicazioni multipiattaforma. Tuttavia, i componenti CLX utilizzano una libreria Qt.

Gli argomenti trattati in questa sezione includono:

- Introduzione alla grafica
- Utilizzo del canvas
- Operazioni con le immagini
- Le bitmap fuori schermo
- Risposta alle modifiche

Introduzione alla grafica

C++Builder incapsula la GDI di Windows (QT di CLX) in vari modi. L'aspetto più significativo per lo sviluppatore di componenti è il modo in cui il componente mostra la propria immagine sullo schermo. Quando viene chiamata direttamente una funzione GDI, è necessario avere un handle per un contesto di dispositivo, all'interno del quale vengono selezionati gli strumenti di disegno come la penna, il pennello e il

font. Una volta terminata la realizzazione dell'immagine, il programmatore deve personalmente riportare il device context allo stato originario e poi rilasciarlo.

Anziché obbligare ad operare sulla grafica a livelli particolareggiati, C++Builder fornisce un'interfaccia semplice ma molto completa: la proprietà *Canvas* di un componente. Il canvas garantisce di avere sempre disponibile un device context valido, che viene rilasciato non appena si è finito di usarlo. Analogamente, il canvas ha una serie di proprietà specifiche che rappresentano la penna, il pennello e il font correnti.

Poiché il canvas gestisce automaticamente tutte queste risorse, lo sviluppatore di componenti non deve preoccuparsi di creare, selezionare e rilasciare direttamente cose come gli handle di penna. È sufficiente ordinare al canvas di utilizzare un certo tipo di penna e il canvas si occuperà del resto.

Uno dei benefici della gestione delle risorse grafiche da parte di C++Builder sta nel fatto che C++Builder può mettere le risorse nella cache, velocizzando notevolmente le operazioni ripetitive. Per esempio, se si sta creando un'applicazione che ripetutamente genera, utilizza e rilascia un particolare tipo di penna, non è necessario ripetere tutte queste operazioni ogni volta all'interno di un ciclo. C++Builder, infatti, salva le risorse grafiche all'interno della cache; in questo modo non deve ricreare lo strumento, ma si limita a riutilizzarne uno esistente.

Un esempio di quanto detto può essere una applicazione che ha dozzine di schede aperte, con centinaia di controlli. Ognuno di questi controlli può avere una o più proprietà *TFont*. Anche se ciò può generare centinaia o migliaia di istanze di oggetti *TFont*, la maggior parte delle applicazioni tende ad utilizzare solo due o tre handle di font grazie alla cache per i font.

Di seguito sono riportati due esempi che illustrano quanto sia semplice la codifica della grafica con C++Builder. Il primo esempio usa le funzioni GDI standard per disegnare un'ellisse gialla tracciata su sfondo blu nella finestra, secondo una tecnica adottata con altri strumenti di sviluppo. Il secondo usa un canvas per disegnare la stessa ellisse in un'applicazione scritta con C++Builder.

Questo è il codice ObjectWindows:

```
void TMyWindow::Paint(TDC& PaintDC, bool erase, TRect& rect)
{
  HPEN PenHandle, OldPenHandle;
  HBRUSH BrushHandle, OldBrushHandle;
  PenHandle = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
  OldPenHandle = SelectObject(PaintDC, PenHandle);
  BrushHandle = CreateSolidBrush(RGB(255, 255, 0));
  OldBrushHandle = SelectObject(PaintDC, BrushHandle);
  Ellipse(10, 20, 50, 50);
  SelectObject(OldBrushHandle);
  DeleteObject(BrushHandle);
  SelectObject(OldPenHandle);
  DeleteObject(PenHandle);
)
```

Questo è il codice C++Builder che svolge esattamente lo stesso compito:

```
void __fastcall TForm1::FormPaint(TObject *Sender)
```

```
{
  Canvas->Pen->Color = clBlue;
  Canvas->Brush->Color = clYellow;
  Canvas->Ellipse(10, 20, 50, 50);
}
```

Utilizzo del canvas

La classe canvas incapsula controlli grafici a vari livelli, fra cui funzioni ad alto livello per tracciare singole linee, figure e testo, proprietà intermedie per gestire le capacità di disegno del canvas, e, nella VCL, consente l'accesso a basso livello alla GDI di Windows.

La Tabella 50.1 riassume le capacità del canvas.

Tabella 50.1 Riepilogo delle capacità del canvas

Livello	Operazione	Strumenti
Alto	Disegno di linee e figure	Metodi *MoveTo*, *LineTo*, *Rectangle*, e *Ellipse*
	Visualizzazione e dimensionamento del testo	Metodi *TextOut*, *TextHeight*, *TextWidth*, e *TextRect*
	Riempimento dell'area	Metodi *FillRect* e *FloodFill*
Intermedio	Personalizzazione di testo e grafica	Proprietà Pen, *Brush*, e *Font*
	Trattamento dei pixel	Proprietà *Pixels*.
	Copia e fusione di immagini	Metodi Draw, *StretchDraw*, *BrushCopy*, e *CopyRect*; proprietà *CopyMode*
Basso	Chiamata di funzioni della GDI di Windows	Proprietà *Handle*

Per informazioni più dettagliate sulle classi canvas e sui relativi metodi e proprietà, consultare la Guida in linea.

Operazioni con le immagini

La maggior parte delle operazioni grafiche in C++Builder riguarda il disegno di componenti e di schede direttamente su canvas. C++Builder consente anche la gestione di immagini separate dai componenti, come bitmap, metafile e icone e gestisce automaticamente le tavolozze dei colori.

Ci sono tre elementi importanti da tener presenti quando si lavora con le immagini in C++Builder:

- Uso di classi picture, graphic o canvas
- Caricamento e memorizzazione di grafici
- Gestione delle tavolozze

Uso di classi picture, graphic o canvas

In C++Builder ci sono tre tipi di classi che permettono di gestire la grafica:

- Un *canvas*, che rappresenta una superficie disegnabile punto per punto sulla quale viene fatta apparire una scheda, un controllo grafico, una bitmap o una stampante. Un canvas è sempre una proprietà di un altro elemento e mai una classe a se stante.
- Un elemento *graphic* rappresenta un'immagine grafica dello stesso tipo di quelle che si trovano normalmente all'interno di un file o di una risorsa, come una bitmap, un'icona o un metafile. C++Builder definisce le classi *TBitmap*, *TIcon*, e *TMetafile* (solo per VCL), tutte e tre derivate da una generica classe *TGraphic*. Lo sviluppatore può anche definire proprie classi grafiche. Definendo un'interfaccia minima standard per tutti gli elementi grafici, Tgraphic mette a disposizione delle applicazioni un semplice meccanismo per operare facilmente con differenti tipi di elementi grafici.
- Un elemento *picture*, che è un contenitore di un elemento grafico, nel senso che può contenere una qualsiasi classe grafica. Per esempio, un elemento di tipo *TPicture* può contenere una bitmap, un'icona, un metafile o un tipo di grafico definito dall'utente, e un'applicazione può accedere a tutti questi elementi nello stesso modo attraverso la classe *picture*. Per esempio, i controlli immagine hanno una proprietà di nome *Picture*, di tipo *TPicture*, che consente al controllo di mostrare indifferentemente un'icona o un altro tipo di immagine.

È bene tenere presente che una classe picture ha sempre un elemento graphic, che a sua volta potrebbe avere un canvas. (L'unico elemento grafico standard che ha un canvas è *TBitmap*). Di solito, quando si opera con un'immagine, si lavora solamente con le parti della classe grafica esposte tramite *TPicture*. Se serve poter accesso a specifici particolari della stessa classe grafica, si può fare riferimento alla proprietà dell'immagine.

Caricamento e memorizzazione di grafici

Gli elementi picture e graphic di C++Builder possono caricare le proprie immagini da un file e salvarle nello stesso file o in un altro. Il caricamento o il salvataggio delle immagini può essere effettuato in qualsiasi momento.

È anche possibile caricare e salvare immagini utilizzando una sorgente Qt MIME o un oggetto stream nel caso si stiano creando componenti CLX.

Per caricare un'immagine da un file si deve chiamare il metodo *LoadFromFile* del componente picture. Per salvare un'immagine in un file, si deve chiamare il metodo *SaveToFile* del componente picture.

LoadFromFile e *SaveToFile* accettano come unico parametro il nome del file. *LoadFromFile* utilizza l'estensione del nome del file per determinare il tipo di oggetto grafico che dovrà creare e caricare. *SaveToFile* salva qualsiasi tipo di file purché appropriato al tipo di oggetto grafico che sta salvando.

Per caricare una bitmap all'interno di un controllo grafico, per esempio, si passa il nome di un file bitmap al metodo *LoadFromFile* del controllo:

```
void __fastcall TForm1::FormCreate(TObject *Sender)
{
  Image1->Picture->LoadFromFile("c:\\windows\\athena.bmp");
}
```

L'oggetto picture riconosce l'estensione .bmp come estensione standard per i file bitmap, pertanto crea il proprio grafico come *TBitmap* e quindi chiama il metodo *LoadFromFile* del grafico. Poiché il grafico è una bitmap, carica l'immagine dal file sotto forma di bitmap.

Gestione delle tavolozze

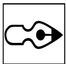
Nel caso di componenti della VCL e della CLX, quando l'esecuzione del programma avviene su dispositivi basati su tavolozza (tipicamente, in modalità video a 256 colori), C++Builder controlla automaticamente la realizzazione della tavolozza di supporto. Cioè se si ha un controllo che ha una tavolozza, si possono utilizzare i due metodi ereditati da *TControl* per controllare come Windows utilizza la tavolozza.

Il supporto delle tavolozza per i controlli presenta due aspetti:

- Specifica di una tavolozza per un controllo
- Risposta alle modifiche della tavolozza

Molti controlli non hanno bisogno di una tavolozza, ma i controlli che contengono immagini grafiche "molto colorate" (come un controllo image) possono aver bisogno di interagire con Windows e con il driver dello schermo per assicurare un aspetto appropriato. Windows fa riferimento a questo processo definendolo *realizing* delle tavolozze.

Il *realizing* delle tavolozze è il processo che garantisce che la finestra frontale utilizzi la tavolozza completa, mentre quelle di sfondo ne utilizzino il massimo possibile, mappando poi gli altri colori a quelli più simili disponibili nella tavolozza "effettiva". Quando le finestre vengono spostate l'una di fronte all'altra, Windows esegue nuovamente il *realizing*.



Anche C++Builder non fornisce un supporto specifico per creare o gestire le tavolozze, se non nel caso delle bitmap. Se si dispone di un handle di tavolozza, comunque, i controlli C++Builder sono in grado di gestirlo.

Specifica di una tavolozza per un controllo

Per specificare la tavolozza per un controllo della VCL e della CLX, ridefinire il metodo *GetPalette* del controllo affinché restituisca l'handle della tavolozza.

Quando la tavolozza di un controllo viene specificata esplicitamente, avvengono due cose:

- L'applicazione viene informata che la tavolozza del controllo deve essere sottoposta al processo di *realizing*.
- La tavolozza da usare nel *realizing* viene esplicitata.

Risposta alle modifiche della tavolozza

Se un controllo della VCL e della CLX specifica una tavolozza ridefinendo *GetPalette*, C++Builder si occupa automaticamente di rispondere ai messaggi di Windows relativi alla tavolozza. Il metodo che gestisce i messaggi relativi alle tavolozza è *PaletteChanged*.

Lo scopo primario di *PaletteChanged* è determinare se il *realizing* della tavolozza del controllo deve avvenire in foreground o in background. Windows gestisce il *realizing* delle tavolozza facendo in modo che la finestra frontale abbia una tavolozza di foreground e che le altre finestre dispongano di tavolozze di background. C++Builder migliora questo trattamento poiché provvede al *realizing* delle tavolozze dei controlli che si trovano all'interno di una stessa finestra secondo la sequenza di accesso tramite il tasto TAB. L'unico caso in cui potrebbe essere necessario ridefinire questo comportamento predefinito è quando si desidera che un controllo, che non è il primo nella sequenza, abbia la tavolozza di foreground.

Le bitmap fuori schermo

Durante il disegno di immagini grafiche complesse, una comune tecnica nella programmazione grafica consiste nel creare una bitmap esternamente allo schermo, di tracciare l'immagine sulla bitmap e di copiare infine l'immagine completa dalla bitmap alla destinazione finale sullo schermo. L'utilizzo di questa tecnica riduce lo sfarfallio che sarebbe altrimenti causato dalle ripetute operazioni di disegno direttamente sullo schermo.

La classe bitmap di C++Builder, usata per rappresenta immagini bitmap in risorse e file, può operare anch'essa come un'immagine fuori schermo.

I due aspetti principali relativi alle operazioni con bitmap fuori schermo sono:

- Creazione e gestione di bitmap fuori schermo.
- Copia di immagini bitmap.

Creazione e gestione di bitmap fuori schermo

Quando si crea un'immagine grafica complessa, è consigliabile non disegnarla direttamente sul canvas che appare sullo schermo. Anziché disegnare sul canvas della scheda o del controllo, si può costruire un oggetto bitmap, disegnare sul canvas dell'oggetto, e infine copiare l'immagine così completata nel canvas sullo schermo. L'uso più comune di una bitmap fuori schermo avviene nel metodo *Paint* di un controllo grafico.

Per un esempio di come disegnare un'immagine complessa su una bitmap fuori schermo, esaminare il codice sorgente del controllo Gauge presente nella pagina Samples della Component palette. Il controllo gauge traccia le varie forme e il testo su una bitmap fuori schermo prima di copiarle sullo schermo. Il codice sorgente del controllo è reperibile nel file Cgauges.cpp presente nella subdirectory Examples\Controls\Source.

Copia di immagini bitmap

C++Builder consente di usare quattro modi differenti per copiare le immagini da un canvas all'altro. A seconda dell'effetto che si vuole ottenere, occorre chiamare un metodo differente.

La Tabella 50.2 riassume i metodi per la copia di immagini negli oggetti canvas.

Tabella 50.2 Metodi per la copia di immagini

Per creare questo effetto	Chiamare questo metodo
Copiare un intero grafico.	Draw
Copiare e ridimensionare un grafico.	StretchDraw
Copiare parte di un canvas.	CopyRect
Copiare una bitmap con operazioni raster.	BrushCopy (VCL)
Copiare un grafico più volte affiancandolo fino a riempire un'area.	TiledDraw(CLX)

Risposta alle modifiche

Tutti gli oggetti grafici, incluso i canvas e i relativi oggetti posseduti (penne, pennelli, e font), possiedono eventi predefiniti per rispondere alle modifiche dell'oggetto. Usando questi eventi, si può fare in modo che i componenti (o le applicazioni che li usano) rispondano alle modifiche ridisegnando le proprie immagini.

Rispondere alle modifiche negli oggetti grafici è particolarmente importante se li si rende pubblici come parte dell'interfaccia di progettazione dei componenti. L'unico modo per assicurarsi che l'aspetto del componente in fase di progetto coincida con le proprietà impostate nell' Object Inspector è quello di rispondere alle modifiche degli oggetti.

Per rispondere ai cambiamenti in un oggetto grafico, assegnare un metodo all'evento *OnChange* della classe.

Il componente shape rende pubbliche le proprietà che rappresentano la penna e il pennello usati per disegnarne il contorno. Il costruttore del componente assegna un metodo all'evento *OnChange* di ciascuno di essi, facendo in modo che il componente si ridisegni automaticamente a video nel caso vengano modificati la penna o il pennello. Sebbene il componente shape sia scritto in Object Pascal, quella che segue è la traduzione in linguaggio C++ del componente shape con un nuovo nome, *TMyShape*.

Questa è la dichiarazione della classe nel file header:

```
class PACKAGE TMyShape : public TGraphicControl
{
private:
protected:
public:
    virtual __fastcall TMyShape(TComponent* Owner);
__published:
```

```
    TPen *FPen;
    TBrush *FBrush;
    void __fastcall StyleChanged(TObject *Sender);
};
```

Questo è il codice nel file .CPP:

```
__fastcall TMyShape::TMyShape(TComponent* Owner)
  : TGraphicControl(Owner)
{
  Width = 65;
  Height = 65;
  FPen = new TPen;
  FPen->OnChange = StyleChanged;
  FBrush = new TBrush;
  FBrush->OnChange = StyleChanged;
}

void __fastcall TMyShape::StyleChanged(TObject *Sender)
{
  Invalidate();
}
```

Capitolo

51

Gestione dei messaggi e delle notifiche di sistema

I componenti spesso devono rispondere alle notifiche dal sistema operativo sottostante. Il sistema operativo informa l'applicazione di accadimenti, come ad esempio, ciò che fa l'utente con il mouse e con la tastiera. Anche alcuni controlli generano delle notifiche, come i risultati delle azioni dell'utente quale, ad esempio, la selezione di un elemento in una casella di riepilogo. La VCL e la CLX gestiscono già la maggior parte delle comuni notifiche. È comunque possibile che vi sia la necessità di scrivere un codice proprio per gestire tali notificazioni.

Nella VCL, le notifiche arrivano sotto forma di *messages*. Questi messaggi possono provenire da una qualsiasi sorgente, inclusi Windows, i componenti della VCL e i componenti che sono statidefiniti. Vi sono tre aspetti da considerare quando si opera con i messaggi:

- Il sistema di gestione dei messaggi
- Modifica della gestione dei messaggi
- Creazione di nuovi gestori di messaggio

In CLX, le notifiche arrivano sotto forma di segnali e di eventi di sistema invece che sotto forma di messaggi di Windows. Per i dettagli su come operare con le notifiche di sistema nella CLX, vedere "Risposta alle notifiche di sistema utilizzando la CLX" a pagina 51-11.

Il sistema di gestione dei messaggi

Tutte le classi della VCL dispongono di un meccanismo incorporato per la gestione dei messaggi, cosiddetti metodi di gestione dei messaggi o gestori dei messaggi. L'idea fondamentale nella gestione dei messaggi è che le classi ricevono messaggi di qualche tipo e li distribuiscono, chiamando un determinato metodo appartenente a

un gruppo, a seconda del messaggio ricevuto.Se non esiste nessun metodo specifico per un determinato messaggio, viene richiamato un gestore predefinito.

Lo schema che segue mostra come funziona il meccanismo di smistamento dei messaggi:



La Visual Component Library definisce un sistema di smistamento dei messaggi che trasforma tutti i messaggi di Windows (compresi quelli definiti dall'utente), diretti a una classe particolare, in chiamate a metodi. Non sarà mai necessario modificare il meccanismo di smistamento dei messaggi. Tutto ciò che occorre fare è creare metodi per gestire i messaggi. Per maggiori informazioni consultare la sezione "Dichiarazione di un nuovo metodo di gestione messaggi" a pagina 51-8.

Cosa contiene un messaggio di Windows?

Un messaggio di Windows può essere immaginato come una struttura di dati che contiene diversi membri dati utili. Il più importante di questi è rappresentato da un valore di tipo integer che identifica il messaggio. Windows definisce molti messaggi, e il file MESSAGES.HPP ne dichiara tutti gli identificatori.

I programmatori Windows sono soliti lavorare con le definizioni di Windows che identificano un messaggio, come, ad esempio, *WM_COMMAND* o *WM_PAINT*. Un programma Windows tradizionale contiene una procedura di finestra che serve come callback per i messaggi generati dal sistema. In questa procedura di finestra esiste solitamente un'istruzione **switch** piuttosto corposa con varie istruzioni case, una per ciascun messaggio che questa finestra è in grado di gestire.

A questa procedura per finestra vengono passate ulteriori informazioni utili mediante due parametri, *wParam* e *lParam*, per "*word parameter*" e "*long parameter*". Spesso, ogni parametro contiene più di una porzione di informazione ed è necessario estrarre la porzione rilevante mediante le macro di Windows, quali *LOWORD* e *HIWORD*. Per esempio, chiamando *HIWORD*(*lParam*) si ricava la word alta di questo parametro.

Originariamente, i programmatori Windows erano costretti a ricordare o a cercare nelle API di Windows quali informazioni erano contenute in ogni parametro. Windows ha recentemente implementato i "*message crackers*" per semplificare la sintassi relativa alla gestione dei messaggi Windows e dei parametri associati. Con i "*message crackers*," anziché utilizzare una vasta istruzione **switch** che analizza tutte le informazioni contenute nei parametri, si può semplicemente associare al messaggio una funzione di gestione. Se si include WINDOWSX.H in un programma Windows standard, risulta disponibile la macro *HANDLE_MSG* e si può scrivere un codice come questo:

```
void MyKeyDownHandler(  HWND hwnd, UINT nVirtKey, BOOL fDown, int CRepeat, UINT flags )
{
  ⋮
}

LRESULT MyWndProc( HWND hwnd, UINT Message, WPARAM wParam, LPARAM lParam )
```

```
{
  switch(  Message )
  {
    HANDLE_MSG( hwnd, WM_KEYDOWN, MyKeyDownHandler );
    ⋮
}
```

L'uso di questo stile di frantumazione dei messaggi rende più palese il fatto che i messaggi vengono smistati a particolari gestori. Inoltre, si possono dare nomi significativi alla lista dei parametri per le proprie funzioni di gestione. In questo modo è più facile comprendere una funzione che accetta un parametro chiamato *nVirtKey*, il quale rappresenta il valore di *wParam* in un messaggio *WM_KEYDOWN*.

Smistamento dei messaggi

Quando un'applicazione crea una finestra, essa registra nel kernel di Windows una procedura di finestra. La procedura di finestra è la routine che gestisce i messaggi per la finestra. Tradizionalmente, nella procedura di finestra c'è un'istruzione **switch** piuttosto corposa che contiene gli elementi di ciascun messaggio che dovrà essere gestito dalla finestra. È bene ricordare che il termine “finestra”, in questa accezione, può far riferimento a qualsiasi cosa presente sullo schermo: ogni finestra, ogni controllo e così via. Ogni volta che si crea un nuovo tipo di finestra, è necessario creare una procedura di finestra completa.

La VCL semplifica lo smistamento dei messaggi in vari modi:

- Ciascun componente eredita un sistema completo per lo smistamento dei messaggi.
- Il sistema di smistamento ha una gestione standard. Il programmatore deve definire solo gli handle per i messaggi ai quali va data una risposta particolare.
- È possibile modificare una piccola parte della gestione dei messaggi e affidarsi ai metodi ereditati per la maggior parte dell'elaborazione.

Il maggior vantaggio di questo sistema di smistamento messaggi risiede nella possibilità di inviare in tutta sicurezza qualsiasi messaggio a qualsiasi componente in qualsiasi momento. Se il componente non ha un gestore per quel messaggio, è la gestione predefinita che se ne occupa, di solito ignorando il messaggio.

Il flusso dei messaggi

La VCL registra un metodo di nome *MainWndProc* come procedura per finestra per ciascun tipo di componente presente in un'applicazione. *MainWndProc* contiene un blocco per la gestione delle eccezioni, che passa la struttura del messaggio da Windows a un metodo virtuale di nome *WndProc* e che gestisce qualunque eccezione chiamando il metodo *HandleException* della classe applicazione.

MainWndProc è un metodo non virtuale che non contiene alcuna gestione per nessun particolare messaggio. Le personalizzazioni avvengono in *WndProc*, dal momento che ogni tipo di componente può ridefinire il metodo per soddisfare le proprie particolari necessità.

I metodi *WndProc* controllano tutte le condizioni particolari che possono influenzare l'elaborazione "catturando" i messaggi non desiderati. Per esempio, mentre vengono trascinati, i componenti ignorano gli eventi di tastiera, pertanto il metodo *WndProc* di *TWinControl* lascia passare gli eventi di tastiera solo se il componente non viene trascinato. Infine, *WndProc* chiama *Dispatch*, un metodo non virtuale ereditato da *TObject*, che stabilisce quale metodo chiamare per gestire il messaggio.

Dispatch utilizza il membro dati *Msg* della struttura del messaggio per stabilire come smistare un particolare messaggio. Se il componente definisce un gestore per quel particolare messaggio, *Dispatch* chiama il metodo. Se il componente non definisce un gestore per quel messaggio, *Dispatch* chiama *DefaultHandler*.

Modifica della gestione dei messaggi

Prima di cambiare la gestione dei messaggi dei propri componenti, accertarsi che questo sia veramente ciò che si desidera. La VCL traduce la maggior parte dei messaggi Windows in eventi che possono essere gestiti sia da chi scrive componenti sia da chi li utilizza. Invece di cambiare il comportamento della gestione dei messaggi, è più probabile che occorra modificare il comportamento della gestione dell'evento.

Per cambiare la gestione dei messaggi nei componenti della VCL, si ridefinisce il metodo che gestisce il messaggio. In determinate circostanze, si può anche impedire che un componente gestisca un messaggio, intercettando il messaggio stesso.

Ridefinizione del metodo di gestione

Per cambiare il modo in cui un componente gestisce un particolare messaggio, si deve ridefinire il metodo che lo gestisce. Se il componente non gestisce già quel particolare messaggio, diventa necessario dichiarare un nuovo metodo per la gestione.

Per ridefinire il metodo di gestione di un messaggio,

1 Dichiarare nella parte protetta della dichiarazione del componente un nuovo metodo nel proprio componente con lo stesso nome del metodo che si vuole ridefinire.

Correlare il metodo al messaggio che esso ridefinisce utilizzando tre macro.

Le macro assumono questa forma:

```
BEGIN_MESSAGE_MAP
  MESSAGE_HANDLER(parameter1, parameter2, parameter3)
END_MESSAGE_MAP
```

Parameter1 è l'indice del messaggio definito da Windows, *parameter2* è il tipo di struttura del messaggio, e *parameter3* è il nome del metodo del messaggio.

È possibile inserire un numero qualunque di macro *MESSAGE_HANDLER* tra le macro *BEGIN_MESSAGE_MAP* e *END_MESSAGE_MAP*.

Per esempio, per ridefinire la gestione del componente del messaggio *WM_PAINT*, si deve ridichiarare il metodo *WMPaint*, e con le tre macro si deve correlare il metodo al messaggio *WM_PAINT*:

```
class PACKAGE TMyComponent : public TComponent
{
protected:
    void __fastcall WMPaint(TWMPaint* Message);

BEGIN_MESSAGE_MAP
  MESSAGE_HANDLER(WM_PAINT, TWMPaint, WMPaint)
END_MESSAGE_MAP(TComponent)
};
```

Uso dei parametri del messagio

Una volta all'interno del metodo di gestione dei messaggi, il componente ha accesso a tutti i parametri della struttura dei messaggi. Poiché il parametro passato al gestore di messaggi è un puntatore, il gestore, se necessario, può modificare il valore dei parametri. L'unico parametro che cambia frequentemente è il valore di ritorno del messaggio: il valore restituito dalla chiamata a *SendMessage* che invia il messaggio.

Poiché nel metodo di gestione dei messaggi il tipo del parametro *Message* varia in funzione del messaggio gestito, per informazioni sui nomi e sul significato dei singoli parametri è necessario fare riferimento alla documentazione relativa ai messaggi di Windows. Se per qualsiasi ragione fosse necessario fare riferimento ai parametri del messaggio usando la vecchia denominazione (*WParam*, *LParam* e così via), è possibile effettuare la conversione di tipo da *Message* al tipo generico *TMessage*, che utilizza quei nomi di parametri.

Intercettazione dei messaggi

In alcune circostanze potrebbe risultare preferibile che i componenti ignorino i messaggi. Vale a dire, si potrebbe impedire che il componente smisti il messaggio al proprio gestore. Per intercettare il messaggio, occorre ridefinire il metodo virtuale *WndProc* .

Per i componenti della VCL, il metodo *WndProc* analizza i messaggi prima di passarli al metodo *Dispatch*, il quale, a sua volta, determina quale metodo usare per gestire il messaggio. Se si ridefinisce *WndProc*, il componente ha la possibilità di filtrare i messaggi prima di smistarli. La ridefinizione di *WndProc* di un controllo derivato da *TWinControl* assomiglia all'esempio seguente:

```
void __fastcall TMyControl::WndProc(TMessage& Message)
{
   // tests to determine whether to continue processing
   if(Message.Msg != WM_LBUTTONDOWN)
      TWinControl::WndProc(Message);
}
```

Il componente *TControl* definisce l'intera serie di messaggi relativi al mouse e li filtra se un utente esegue operazioni di "drag and drop" con i controlli. Ridefinendo *WndProc*, questo compito viene facilitato in due modi:

- È possibile filtrare gruppi di messaggi anziché specificare i gestori per ciascuno di essi.
- È possibile impedire completamente che il messaggio venga smistato, in modo che i gestori non siano mai chiamati.

Qui di seguito viene riportata una parte del metodo *WndProc* per *TControl* così come è stata implementata nella VCL in Object Pascal:

```
procedure TControl.WndProc(var Message: TMessage);
begin
  ⋮
  if (Message.Msg >= WM_MOUSEFIRST) and (Message.Msg <= WM_MOUSELAST) then
    if Dragging then                                    { handle dragging specially }
      DragMouseMsg(TWMMouse(Message))
    else
      ⋮                                                    { handle others normally }
    end;
⋮                                                      { otherwise process normally }
end;
```

Creazione di nuovi gestori di messaggio

Poiché la VCL mette a disposizione i gestori per i messaggi più comuni, il momento più probabile in cui sarà necessario creare nuovi gestori di messaggi, sarà quando si definiranno i propri messaggi. Lavorare con messaggi definiti dall'utente presenta tre aspetti:

- Definizione di un proprio messaggio.
- Dichiarazione di un nuovo metodo di gestione messaggi.
- Invio di messaggi.

Definizione di un proprio messaggio

Una gran quantità di componenti standard definisce messaggi per uso interno. Il motivo più comune che induce alla definizione di messaggi è la necessità di trasmettere informazioni non coperte dai messaggi standard e di notificare i cambiamenti di stato. È possibile definire i propri messaggi nella VCL.

La definizione di un messaggio è un processo in due fasi. Queste sono:

1 Dichiarazione di un identificatore di messaggio.

2 Dichiarazione di un tipo di struttura di messaggio.

Dichiarazione di un identificatore di messaggio

Un identificatore di messaggio è una costante di tipo integer. Windows si riserva i numeri inferiori a 1.024 per proprio uso; pertanto se si vuole dichiarare un messaggio personalizzato, è necessario utilizzare un valore più alto.

La costante *WM_APP* rappresenta il numero iniziale per i messaggi definiti dall'utente. Quando si definiscono gli identificatori di messaggio, si deve utilizzare come numero base *WM_APP*.

È bene tenere presente che alcuni controlli standard di Windows utilizzano messaggi nell'intervallo definito dall'utente. Questi includono le caselle di riepilogo, le caselle di testo, le caselle combinate e i pulsanti di comando. Se si vuole derivare un componente da uno di questi elementi e si vuole definire per esso un nuovo messaggio, bisogna ricordarsi di controllare il file MESSAGES.HPP per vedere quali messaggi Windows ha già definito per quel controllo.

Il codice riportato di seguito mostra due messaggi definiti dall'utente.

```
#define MY_MYFIRSTMESSAGE  (WM_APP + 400)
#define MY_MYSECONDMESSAGE (WM_APP + 401)
```

Dichiarazione di un tipo di struttura di messaggio

Se si vogliono assegnare nomi significativi ai parametri del messaggio, si deve dichiarare per quel messaggio un tipo *message* structure. *Message* structure è il tipo di parametro passato al metodo di gestione messaggi. Se non si utilizzano i parametri di messaggio, o se si vuole utilizzare la notazione vecchio stile (*wParam*, *lParam*, ecc.), si può utilizzare *TMessage*, la *struttura di messaggio* predefinita.

Per dichiarare un tipo message-structure è necessario seguire queste convenzioni:

1. Assegnare un nome al tipo structure dopo il messaggio, anteponendo una *T*.
2. Assegnare al primo membro dati nella struttura il nome *Msg*, di tipo *TMsgParam*.
3. Definire i successivi due byte in modo che corrispondano al parametro *Word*, e gli altri due byte successivi come non utilizzati.

   Oppure

   Definire i successivi quattro byte in modo che corrispondano al parametro *Longint*.
4. Aggiungere un membro dati finale di nome *Result*, di tipo *Longint*.

Ad esempio, questa è la struttura per tutti i messaggi del mouse, *TWMKey*:

```
struct TWMKey
{
  Cardinal Msg;                    // first parameter is the message ID
  Word CharCode;                   // this is the first wParam
  Word Unused;
  Longint KeyData;                 // this is the lParam
  Longint Result;                  // this is the result data member
};
```

Dichiarazione di un nuovo metodo di gestione messaggi

In due casi diventa necessario dichiarare un nuovo metodo di gestione messaggi:

- Quando il componente deve gestire un messaggio di Windows che non è già gestito da uno dei componenti standard.
- Quando è stato definito un messaggio personalizzato che verrà usato dai propri componenti.

Per dichiarare un metodo di gestione messaggi, occorre:

1. Dichiarare il metodo nella sezione **protected** della dichiarazione della classe del componente utilizzando le macro BEGIN_MESSAGE_MAP ... END_MESSAGE_MAP.
2. Assicurarsi che il metodo restituisca un void.
3. Assegnare un nome al metodo dopo il messaggio da esso gestito, ma senza il carattere di sottolineatura.
4. Passare un puntatore di nome *Message* di tipo message-structure.
5. Mappare il metodo al messaggio utilizzando le macro.
6. All'interno dell'implementazione del metodo del messaggio, scrivere il codice per qualsiasi gestione particolare relativa al componente.
7. Chiamare il gestore del messaggio ereditato.

Ecco, per esempio, la dichiarazione di un gestore di messaggi per un messaggio definito dall'utente di nome *CM_CHANGECOLOR*.

```
#define CM_CHANGECOLOR (WM_APP + 400)

class TMyControl : public TControl
{
protected:
    void __fastcall CMChangeColor(TMessage &Message);

BEGIN_MESSAGE_MAP
  MESSAGE_HANDLER(CM_CHANGECOLOR, TMessage, CMChangeColor)
END_MESSAGE_MAP(TControl)
};

void __fastcall TMyControl::CMChangeColor(TMessage &Message)
{
  Color = Message.LParam;                 // set color from long parameter
  TControl::CMChangeColor(Message);       // call the inherited message handler
}
```

Invio di messaggi

Di solito, un'applicazione manda un messaggio per inviare notifiche di modifiche allo stato o per trasmettere informazioni. Il componente può trasmettere messaggi a tutti i controlli in una scheda, inviare messaggi a un particolare controllo (o all'applicazione stessa) o anche inviare messaggi a se stesso.

Esistono vari modi per inviare un messaggio Windows. Il metodo utilizzato dipende dal tipo di messaggio inviato. I seguenti argomenti descrivono i diversi modi per inviare messaggi Windows.

Trasmissione di un messaggio a tutti i controlli in una scheda

Quando il componente modifica delle impostazioni globali che influiscono su tutti i controlli in una scheda o in altro contenitore, si potrebbe volere inviare un messaggio a quei controlli in modo che essi si possano aggiornare adeguatamente. Non tutti i controlli potrebbero avere bisogno di rispondere alla notifica, ma trasmettendo il messaggio, è possibile informare tutti i controlli che sanno come rispondere e permettere agli altri controlli di ignorare il messaggio.

Per trasmettere un messaggio a tutti i controlli in un altro controllo, utilizzare il metodo *Broadcast*. Prima di trasmettere un messaggio, si riempie una struttura di messaggio con le informazioni che si desidera comunicare. (Per informazioni sulle strutture dei messaggi, vedere “Dichiarazione di un tipo di struttura di messaggio” a pagina 51-7).

```
TMessage Msg;
Msg.Msg = MY_MYCUSTOMMESSAGE;
Msg.WParam = 0;
Msg.LParam = (int)(this);
Msg.Result = 0;
```

Quindi, passare questa struttura di messaggio al genitore di tutti i controlli a cui si desidera inviare la notifica. Questi può essere un qualsiasi controllo nell’applicazione. Ad esempio, può essere il genitore del controllo che si sta scrivendo:

```
Parent->Broadcast(Msg);
```

Può essere la scheda che contiene il controllo:

```
GetParentForm(this)->Broadcast(Msg);
```

Può essere la scheda attiva:

```
Screen->ActiveForm->Broadcast(Msg);
```

Possono anche essere tutte le schede nell’applicazione:

```
for (int i = 0; i < Screen->FormCount; i++)
  Screen->Forms[i]->Broadcast(Msg);
```

Chiamata diretta del gestore del messaggio di un controllo

Talvolta c’è solo un singolo controllo che deve rispondere al messaggio. Se si conosce il controllo che dovrebbe ricevere il messaggio, il modo più semplice e diretto per inviare il messaggio consiste nel chiamare il metodo *Perform* del controllo.

Vi sono due motivi principali per cui si chiama il metodo *Perform* di un controllo:

- Si desidera innescare la stessa risposta che il controllo dà ad un messaggio standard Windows (o ad un altro). Ad esempio, quando un controllo griglia riceve un messaggio di pressione tasti, crea un controllo inline di edit e quindi invia il messaggio di pressione tasti al controllo di edit.

- Il controllo a cui si desidera inviare la notifica potrebbe essere noto, ma si potrebbe non sapere che tipo di controllo è. Poiché non si conosce il tipo del controllo di destinazione, non si può utilizzare nessuno dei suoi metodi specializzati, ma poiché tutti i controlli hanno la capacità di gestire i messaggi è sempre possibile inviare un messaggio. Se il controllo ha un gestore di messaggio per il messaggio che viene inviato, risponderà adeguatamente. Altrimenti, ignorerà il messaggio inviato e restituirà 0.

Per chiamare il metodo *Perform*, non è necessario creare una struttura di messaggio. È necessario solo passare come parametri l'identificativo del messaggio, *WParam* e *LParam*. *Perform* restituisce il risultato del messaggio.

Invio di un messaggio utilizzando la coda dei messaggi Windows

In un'applicazione multithread non è sufficiente chiamare il metodo *Perform* perché il controllo di destinazione è in un thread diverso da quello che è in esecuzione. Tuttavia, utilizzando la coda dei messaggi Windows, è possibile comunicare in tutta sicurezza con altri thread. La gestione dei messaggi si verifica sempre nel thread VCL principale, ma è possibile mandare un messaggio mediante la coda dei messaggi Windows da un qualsiasi thread nell'applicazione. Una chiamata a *SendMessage* è sincrona. Cioè, *SendMessage* non ritorna finché il controllo di destinazione non ha gestito il messaggio, anche se è in un altro thread.

Utilizzare la chiamata API Windows, *SendMessage*, per inviare un messaggio a un controllo utilizzando la coda dei messaggi Windows. *SendMessage* accetta gli stessi parametri del metodo *Perform*, salvo la necessità di identificare il controllo di destinazione passando il suo handle Window. Così, invece di scrivere

```
MsgResult = TargetControl->Perform(MY_MYMESSAGE, 0, 0);
```

si dovrebbe scrivere

```
MsgResult = SendMessage(TargetControl->Handle, MYMESSAGE, 0, 0);
```

Per maggiori informazioni sulla funzione di *SendMessage*, consultare la documentazione MSDN di Microsoft. Per ulteriori informazioni sulla scrittura di più thread che possono essere contemporaneamente in esecuzione, vedere "Coordinamento dei thread" on pagina 11-7.

Invio di un messaggio che non viene eseguito immediatamente

A volte si potrebbe volere inviare un messaggio ma non si sa se sia sicuro che il destinatario del messaggio lo esegua immediatamente. Ad esempio, se il codice che invia un messaggio viene chiamato da un gestore di evento sul controllo di destinazione, si potrebbe volere essere certi che l'esecuzione del gestore di evento sia terminata prima che il controllo esegua il messaggio. È possibile gestire questa situazione a patto che non sia necessario conoscere il risultato del messaggio.

Utilizzare la chiamata API Windows, *PostMessage*, per inviare un messaggio a un controllo e al contempo consentirgli di attendere di aver terminato di gestire tutti gli altri messaggi prima di gestire il vostro. *PostMessage* accetta esattamente gli stessi parametri di *SendMessage*.

Per maggiori informazioni sulla funzione di *PostMessage*, consultare la documentazione MSDN di Microsoft.

Risposta alle notifiche di sistema utilizzando la CLX

Quando si utilizza Windows, il sistema operativo invia le notifiche direttamente all'applicazione e ai controlli che essa contiene utilizzando messaggi Windows. Questo approccio, tuttavia, non è appropriato per la CLX, in quanto la CLX è una libreria multipiattaforma e i messaggi Windows non sono utilizzati in Linux. Per rispondere alle notifiche di sistema CLX utilizza invece una modalità indipendente dalla piattaforma

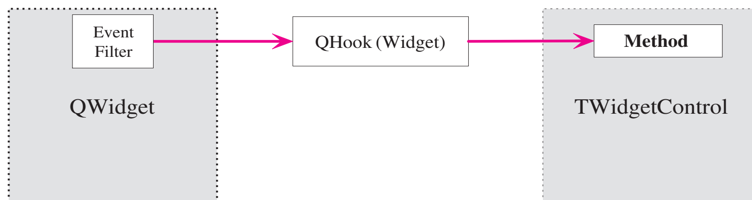
In CLX, la modalità analoga ai messaggi Windows è un sistema di segnali provenienti dallo strato del widget sottostante. Mentre nella VCL i messaggi Windows possono avere origine o dal sistema operativo o dai controlli Windows nativi che la VCL avvolge, lo strato del widget utilizzato dalla CLX fa a una distinzione fra questi due. Se la notifica ha origine da un widget, è detta un segnale. Se la notifica ha origine dal sistema operativo, è detta evento di sistema. Lo strato del widget comunica ai componenti CLX gli eventi di sistema come segnali di tipo *evento*.

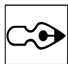
Risposta ai segnali

Lo strato del widget sottostante emette una gamma di segnali, ognuno dei quali rappresenta un tipo diverso di notifica. Questi segnali includono oltre agli eventi di sistema (il segnale di evento) anche le notifiche specifiche del widget che le ha generate. Ad esempio, tutti i widget quando vengono liberati generano un segnale destroyed, i widget trackbar generano un segnale *valueChanged*, i controlli header generano un segnale *sectionClicked*, e così via.

Ogni componente CLX risponde ai segnali provenienti dal proprio widget sottostante assegnando un metodo come gestore del segnale. Ciò viene fatto utilizzando un oggetto hook speciale, associato al widget sottostante. L'oggetto hook è un oggetto leggero che in realtà è solo una raccolta di puntatori a metodo, ognuno dei quali è specifico per un particolare segnale. Quando un metodo del componente CLX è stato assegnato all'oggetto hook come gestore di un segnale specifico, ogni volta che il widget genera il segnale specifico, viene chiamato il metodo sul componente CLX. Questo comportamento è illustrato nella Figura 51.1.

Figura 51.1 Instradamento del segnale

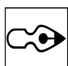




I metodi di ogni oggetto hook sono dichiarati nella unit Qt. Controllare il file qt.hpp per vedere i metodi disponibili per un determinato oggetto hook. I metodi sono semplificati in routine globali con nomi che riflettono l'oggetto hook a cui essi appartengono. Ad esempio, tutti i metodi sull'oggetto hook associato al widget applicazione (QApplication) iniziano ' QApplication_hook'. Questa semplificazione è necessaria in modo che l'oggetto CLX di Object Pascal possa accedere ai metodi dell'oggetto hook di C++.

Assegnazione dei gestori di segnale custom

Molti controlli CLX assegnano già metodi per gestire segnali dal widget sottostante. Di solito, questi metodi sono privati e non virtuali. Pertanto, se si desidera scrivere un proprio metodo per rispondere a un segnale, è necessario assegnare il metodo all'oggetto hook associato al widget. Per fare ciò, si deve ridefinire il metodo *HookEvents*.



Se il segnale a cui si desidera rispondere è una notifica di un evento di sistema, non si deve utilizzare una ridefinizione del metodo *HookEvents*. Per i dettagli su come rispondere agli eventi di sistema, vedere "Risposta a eventi di sistema" nel prosieguo del capitolo.

Nella ridefinizione del metodo *HookEvents*, dichiarare una variabile di tipo *TMethod*. Quindi, per ogni metodo che si desidera assegnare come gestore di segnale all'oggetto hook, fare quanto segue:

1. Inizializzare la variabile del tipo *TMethod* per rappresentare un gestore di metodo per il segnale.
2. Assegnare questa variabile all'oggetto hook. È possibile accedere all'oggetto hook utilizzando la proprietà *Hooks* che il componente eredita da *THandleComponent* o *TWidgetControl*.

Nella ridefinizione, chiamare sempre il metodo ereditato *HookEvents* in modo da connettere anche i gestori di segnale assegnati dalle classi di base.

Il codice seguente è una traduzione del metodo *HookEvents* di *TTrackBar*. Illustra come ridefinire il metodo *HookEvents* per aggiungere dei gestori custom di segnale.

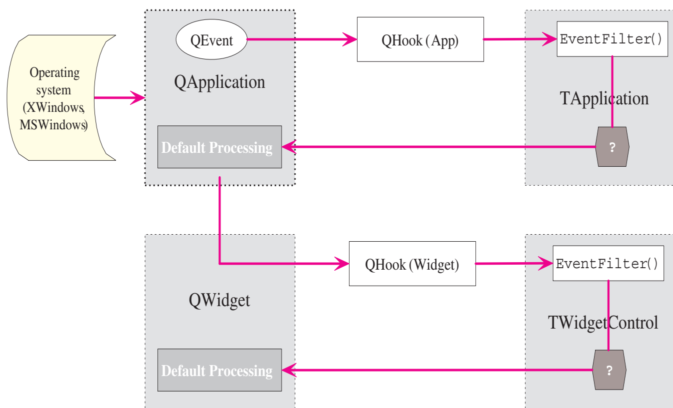
```
virtual void __fastcall  TTrackBar::HookEvents(void)
{
  TMethod Method;
  // initialize Method to represent a handler for the QSlider valueChanged signal
  // ValueChangedHook is a method of TTrackBar that responds to the signal.
  QSlider_valueChanged_Event(Method) = @ValueChangedHook;
  // Assign Method to the hook object. Note that you can cast Hooks to the
  // type of hook object associated with the underlying widget.
  QSlider_hook_hook_valueChanged(dynamic_cast<QSlider_hookH>(Hooks), Method);
  // Repeat the process for the sliderMoved event:
  QSlider_sliderMoved_Event(Method) := @ValueChangedHook;
  QSlider_hook_hook_valueChanged(dynamic_cast<QSlider_hookH>(Hooks), Method);
  // Call the inherited method so that inherited signal handlers are hooked up:
  TWidgetControl::HookEvents();
}
```

Risposta a eventi di sistema

Quando lo strato del widget riceve una notifica di evento dal sistema operativo, genera uno speciale oggetto evento (*QEvent* o uno dei suoi discendenti) per rappresentare l'evento. L'oggetto evento contiene informazioni a sola lettura sull'evento che si è verificato. Il tipo dell'oggetto evento indica il tipo di evento che si è verificato.

Lo strato del widget notifica al componente CLX gli eventi di sistema utilizzando uno speciale segnale di tipo evento. Esso passa l'oggetto *QEvent* al gestore di segnale dell'evento. L'elaborazione del segnale di evento è un po' più complicata rispetto all'elaborazione di altri segnali in quanto arriva prima all'oggetto applicazione. Ciò significa che un'applicazione ha due opportunità per rispondere a un evento di sistema: una volta a livello dell'applicazione (*TApplication*) e una volta a livello del singolo componente (il discendente *TWidgetControl* o *THandleComponent*). Tutte queste classi (*TApplication*, *TWidgetControl* e *THandleComponent*) assegnano già dallo strato del widget un gestore di segnale per il segnale evento. Cioè, tutti gli eventi di sistema vengono indirizzati automaticamente al metodo *EventFilter*, che ha un ruolo simile al metodo *WndProc* dei controlli VCL. La gestione degli eventi di sistema è illustrata nella Figura 51.2.

Figura 51.2 Instradamento dell'evento di sistema



EventFilter gestisce la maggior parte delle notifiche di sistema comunemente utilizzate, traducendole negli eventi che sono introdotti dalle classi di base del componente. Così, ad esempio, il metodo *EventFilter* di *TWidgetControl* risponde agli eventi del mouse (*QMouseEvent*) generando gli eventi *OnMouseDown*, *OnMouseMove* e *OnMouseUp*, agli eventi di tastiera (*QKeyEvent*) generando gli eventi *OnKeyDown*, *OnKeyPress*, *OnKeyString* e *OnKeyUp*, e così via.

Eventi comunemente utilizzati

Il metodo *EventFilter* di *TWidgetControl* gestisce molte delle comuni notifiche di sistema chiamando i metodi protetti introdotti in *TControl* o in *TWidgetControl*. La maggior parte di questi metodi è virtuale, in modo che sia possibile ridefinirli durante la scrittura di componenti e implementare le risposte agli eventi di sistema.

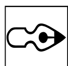
Quando si ridefiniscono questi metodi, non è necessario preoccuparsi di operare con l'oggetto evento o (nella maggior parte dei casi) con un qualunque altro oggetto presente nello strato del widget sottostante.

Quando si desidera che il componente CLX risponda alle notifiche di sistema, è bene per prima cosa controllare se esiste già un metodo protetto in grado di rispondere alla notifica. È possibile controllare la documentazione di *TControl* o di *TWidgetControl* (e qualsiasi altra classe base da cui si deriva il componente) per vedere se esiste un metodo protetto che risponda all'evento a cui si è interessati. La Tabella 51.1 elenca i metodi protetti più comunemente utilizzati da *TControl* e da *TWidgetControl* che è possibile utilizzare.

Tabella 51.1 Metodi protetti di TWidgetControl per la risposta ad eventi di sistema

Metodo	Descrizione
BeginAutoDrag	Chiamato quando l'utente fa clic sul tasto sinistro del mouse se il controllo ha un *DragMode* impostato a *dmAutomatic*.
Click	Chiamato quando l'utente rilascia il tasto del mouse sul controllo.
DblClick	Chiamato quando l'utente fa doppio clic con il mouse sul controllo.
DoMouseWheel	Chiamato quando l'utente gira la rotella del mouse.
DragOver	Chiamato quando l'utente trascina il cursore del mouse sul controllo.
KeyDown	Chiamato quando l'utente preme un tasto mentre il controllo ha il fuoco.
KeyPress	Chiamato dopo *KeyDown* se *KeyDown* non gestisce la pressione del tasto.
KeyString	Chiamato quando l'utente preme un tasto nel caso in cui il sistema utilizzi un sistema di caratteri multibyte.
KeyUp	Chiamato quando l'utente rilascia un tasto mentre il controllo ha il fuoco.
MouseDown	Chiamato quando l'utente fa clic sul tasto del mouse sul controllo.
MouseMove	Chiamato quando l'utente sposta il cursore del mouse sul controllo.
MouseUp	Chiamato quando l'utente rilascia il tasto del mouse sul controllo.
PaintRequest	Chiamato quando il sistema deve ridisegnare il controllo.
WidgetDestroyed	Chiamato quando si distrugge un widget che è alla base di un controllo.

Nella ridefinizione, chiamare il metodo ereditato in modo che qualsiasi processo predefinito possa ancora rispondere ai segnali.



Oltre ai metodi che rispondono agli eventi di sistema, i controlli includono molti metodi simili che hanno origine con *TControl* o *TWidgetControl* per notificare al controllo i diversi eventi. Benché questi non rispondano a eventi di sistema, essi svolgono gli stessi compiti di molti messaggi Windows che vengono inviati ai controlli VCL. La Tabella 51.1 elenca alcuni di questi metodi.

Tabella 51.2 Metodi protetti di TWidgetControl per la risposta ad eventi dai controlli

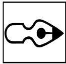
Metodo	Descrizione
BoundsChanged	Chiamato quando il controllo viene ridimensionato.
ColorChanged	Chiamato quando viene modificato il colore del controllo.
CursorChanged	Chiamato quando si modifica la forma del cursore. Il cursore del mouse assume questa forma quando è su questo widget.

Tabella 51.2 Metodi protetti di TWidgetControl per la risposta ad eventi dai controlli (continua)

Metodo	Descrizione
EnabledChanged	Chiamato quando un'applicazione modifica lo stato enabled di una finestra o di un controllo.
FontChanged	Chiamato quando si modifica la raccolta di risorse dei font.
PaletteChanged	Chiamato quando cambia la tavolozza del widget.
ShowHintChanged	Chiamato quando si visualizzano o si nascondono i suggerimenti di un controllo.
StyleChanged	Chiamato quando vengono cambiati gli stili GUI della finestra o del controllo.
TabStopChanged	Chiamato quando si modifica la sequenza di tabulazione sulla scheda.
TextChanged	Chiamato quando il testo del controllo viene modificato.
VisibleChanged	Chiamato quando si nasconde o visualizza un controllo.

Ridefinizione del metodo EventFilter

Se si desidera rispondere a una notifica di evento e non c'è alcun metodo protetto per quell'evento che è possibile ridefinire, è possibile ridefinire lo stesso metodo *EventFilter*. Nella ridefinizione, controllare il tipo del parametro *Event* del metodo *EventFilter* ed eseguire la propria particolare elaborazione quando rappresenta il tipo di notifica a cui si desidera rispondere. È possibile evitare un'ulteriore elaborazione della notifica dell'evento facendo in modo che il metodo *EventFilter* restituisca **true**.



Vedere la documentazione Qt della TrollTech per i dettagli sui vari tipi di oggetti *QEvent*.

Il codice seguente è una traduzione del metodo *EventFilter* di *TCustomControl*. Illustra come ottenere il tipo di evento dall'oggetto *QEvent* quando si ridefinisce *EventFilter*. Notare che, benché non sia visibile nell'esempio, una volta che si è identificato il tipo di evento, è possibile convertire il tipo dell'oggetto *QEvent* in un appropriato discendente specializzato di *QEvent* (come *QMouseEvent*).

```
virtual bool __fastcall  TCustomControl::EventFilter(Qt::QObjectH* Sender, Qt::QEventH*
Event)
{
  bool retval = TWidgetControl::EventFilter(Sender, Event);
  switch (QEvent_type(Event))
  {
    case QEventType_Resize:
    case QEventType_FocusIn:
    case QEventType_FocusOut:
      UpdateMask();
  }
  return retval;
}
```

Generazione di eventi Qt

Analogamente al modo in cui un controllo VCL può definire e inviare messaggi Windows custom, è possibile fare in modo che il controllo CLX definisca e generi eventi Qt di sistema. Il primo passo consiste nel definire un ID univoco per l'evento

(simile al modo in cui è necessario definire un ID di messaggio quando si definisce un messaggio Windows custom):

```
static const MyEvent_ID = (int) QCLXEventType_ClxUser + 50;
```

Nel codice in cui si desidera generare l'evento, si utilizza la funzione *QCustomEvent_create* (dichiarata in Qt.hpp) per creare un oggetto evento con il nuovo ID di evento. Un secondo parametro facoltativo permette di fornire all'oggetto evento un valore che è un puntatore alle informazioni che si vogliono associare all'evento:

```
QCustomEventH *MyEvent = QCustomEvent_create(MyEvent_ID, this);
```

Una volta che si è creato l'oggetto evento, è possibile registrarlo chiamando il metodo *QApplication_postEvent*:

```
QApplication_postEvent(Application->Handle, MyEvent);
```

Affinché qualsiasi componente risponda a questa notifica, è necessario solo ridefinirne il metodo *EventFilter*, controllando un tipo di evento *MyEvent_ID*. Il metodo *EventFilter* può acquisire i dati forniti al costruttore chiamando il metodo *QCustomEvent_data*, dichiarato in Qt.hpp.

Capitolo

52

Come rendere disponibili i componenti in fase di progetto

Questo capitolo descrive le operazioni necessarie per rendere disponibili nell'IDE i componenti creati. Rendere disponibile i propri componenti in fase di progetto è un'operazione che richiede diverse fasi:

- Registrazione di componenti
- Aggiunta di bitmap alla tavolozza
- Preparazione di un file di Guida per il componente
- Aggiunta di editor di proprietà
- Aggiunta di editor di componenti
- Compilazione dei componenti in package

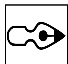
Non tutte queste operazioni valgono per tutti i componenti. Per esempio, se non si definiscono nuovi eventi o proprietà, non occorre predisporre una specifica Guida all'uso. Le uniche operazioni sempre necessarie sono la registrazione e la compilazione.

Una volta che i propri componenti sono stati registrati e compilati in package, essi possono essere distribuiti ad altri sviluppatori e installati nell'IDE. Per maggiori informazioni sull'installazione dei packages nell'IDE, consultare il paragrafo "Installazione di package di componenti" a pagina 15-6.

Registrazione di componenti

La registrazione si fonda sulle unit e pertanto, se si creano molti componenti in una singola unit, è possibile registrarli tutti in una volta.

Per registrare un componente, aggiungere una funzione *Register* nel file .CPP della unit. All'interno della funzione *Register* si registrano i componenti e si determina dove installarli nella Component palette.



Se si crea un componente scegliendo Component | New Component nell'IDE, il codice necessario per la sua registrazione viene aggiunto automaticamente.

I passi per registrare manualmente un componente sono:

- Dichiarazione della funzione Register
- Scrittura della funzione Register

Dichiarazione della funzione Register

La registrazione comporta la scrittura nel file .CPP della unit di un'unica funzione, il cui nome deve essere *Register*. La funzione *Register* deve esistere all'interno di un namespace. Il namespace è il nome del file in cui è contenuto il componente con tutte le lettere in minuscolo eccetto la prima.

Il codice seguente mostra come viene implementata la funzione *Register* all'interno di un namespace. Il namespace viene chiamato Newcomp, mentre il file viene chiamato Newcomp.CPP:

```
namespace Newcomp
{
  void __fastcall PACKAGE Register()
  {
  }
}
```

Dall'interno della funzione *Register*, chiamare *RegisterComponents* per ogni componente che si vuole aggiungere alla Component palette. Se la combinazione del file header e del file .CPP contiene diversi componenti, si può registrarli tutti in un'unica operazione. La macro PACKAGE viene espansa in un'istruzione che consente di importare e di esportare classi.

Scrittura della funzione Register

All'interno della funzione *Register* di una unit contenente componenti, si devono registrare tutti i componenti che si intende aggiungere alla Component palette. Se la unit contiene diversi componenti, è possibile registrarli contemporaneamente.

Per registrare un componente, chiamare la funzione *RegisterComponents* una volta per ogni pagina della Component palette alla quale si vogliono aggiungere componenti. *RegisterComponents* richiede tre cose importanti:

1. Specifica dei componenti
2. Specifica della pagina della tavolozza
3. Uso della funzione RegisterComponents

Specifica dei componenti

All'interno della funzione *Register* dichiarare un array vuoto di tipo *TComponentClass*, destinato a contenere l'array di componenti da installare. La sintassi è più o meno questa:

```
TMetaClass classes[1] = {__classid(TNewComponent)};
```

In questo caso, l'array di classi contiene solo un componente, ma è possibile aggiungervi tutti i componenti che si vogliono registrare. Per esempio, il codice seguente colloca due componenti nell'array:

```
TMetaClass classes[2] =
                    {__classid(TNewComponent), __classid(TAnotherComponent)};
```

Un altro modo per aggiungere un componente all'array consiste nell'assegnarlo in un'istruzione separata. Le seguenti istruzioni aggiungono all'array gli stessi due componenti dell'esempio precedente:

```
TMetaClass classes[2];
classes[0] = __classid(TNewComponent);
classes[1] = __classid(TAnotherComponent);
```

Specifica della pagina della tavolozza

Il nome della pagina della tavolozza è una AnsiString. Se il nome che viene dato alla pagina della tavolozza non esiste ancora, C++Builder crea una nuova pagina con quel nome. C++Builder registra i nomi delle pagine standard in una lista di stringhe, in modo che le versioni internazionali del prodotto possono chiamare le pagine nelle lingue locali. Se si vuole installare un componente in una pagina standard, si deve prima recuperare la stringa per il nome della pagina chiamando la funzione *LoadStr* e passando come argomento la costante che rappresenta la risorsa stringa per quella pagina, come *srSystem* per la pagina System.

Uso della funzione RegisterComponents

Dall'interno della funzione *Register* chiamare *RegisterComponents* per registrare i componenti nell'array delle classi. *RegisterComponents* è una funzione che richiede tre parametri: il nome di una pagina della Component palette, l'array delle classi dei componenti e l'indice dell'ultima voce dell'array.

La seguente funzione *Register*, disponibile nel file NEWCOMP.CPP, registra un componente di nome *TMyComponent* e lo colloca in una pagina della tavolozza Component di nome Miscellaneous:

```
namespace Newcomp
{
  void __fastcall PACKAGE Register()
  {
      TMetaClass classes[1] = {__classid(TMyComponent)};
      RegisterComponents("Miscellaneous", classes, 0);
  }
}
```

Si noti che il terzo argomento della chiamata *RegisterComponents* è 0, che rappresenta l'indice dell'ultima voce dell'array delle classi (la dimensione dell'array meno 1)).

È possibile registrare contemporaneamente diversi componenti nella stessa pagina o registrare i componenti in pagine differenti, come mostra il codice seguente:

```
namespace Mycomps
{
  void __fastcall PACKAGE Register()
  {
```

```
        // declares an array that holds two components
        TMetaClass classes1[2] = {__classid(TFirst), __classid(TSecond)};
        // adds a new palette page with the two components in the classes1 array
        RegisterComponents("Miscellaneous", classes1, 1);
        // declares a second array
        TMetaClass classes2[1];
        // assigns a component to be the first element in the array
        classes2[0]  = __classid(TThird);
        // adds the component in the classes2 array to the Samples page
        RegisterComponents("Samples", classes2, 0);
    }
}
```

Nell'esempio vengono dichiarati due array, classes1 e classes2. Nella prima chiamata a *RegisterComponents* l'array classes1 ha 2 voci, per cui il terzo argomento è l'indice della seconda voce, che è 1. Nella seconda chiamata a *RegisterComponents* l'array classes2 ha un elemento, per cui il terzo argomento è 0.

Aggiunta di bitmap alla tavolozza

Ogni componente ha bisogno di una bitmap per rappresentare il componente nella Component palette. Se non si specifica una propria bitmap, C++Builder ne utilizza una predefinita.

Poiché le tavolozze sono necessarie soltanto in fase di progetto, non occorre compilarle nella unit di compilazione del componente. Devono essere fornite come una file risorsa di Windows con lo stesso nome del file .CPP, ma con estensione .DCR (acronimo di Dynamic Component Resource). Questo file risorsa può essere creato utilizzando l'editor Image in C++Builder. Ogni bitmap deve essere di 24 pixel quadrati.

Per ogni componente che si vuole installare, occorre indicare un file bitmap della tavolozza, all'interno del quale si deve fornire una bitmap per ogni componente che si registra. L'immagine bitmap ha lo stesso nome della classe del componente. È bene tenere il file bitmap della tavolozza nella stessa directory in cui si trova il file compilato, in modo che C++Builder possa trovare le bitmap quando installa i componenti nella Component palette.

Ad esempio, se si vuole creare un componente di nome *TMyControl*, occorre creare un file risorsa .DCR o .RES contenga una bitmap di nome TMYCONTROL. I nomi dei file risorsa possono essere scritti in lettere maiuscole o minuscole, ma per convenzione, sono generalmente in maiuscolo.

Preparazione di un file di Guida per il componente

Se si seleziona un componente standard su una scheda, o una proprietà o un evento nell'Object Inspector, premendo il tasto *F1* si richiama la Guida relativa a quel determinato elemento. Creando opportuni file di Guida, si può fornire agli sviluppatori lo stesso tipo di documentazione per i propri componenti.

È possibile fornire un piccolo file di guida che descriva i propri componenti. In questo modo il proprio file diviene parte della Guida utente del sistema C++Builder.

Per informazioni su come comporre un file di Guida da utilizzare con i componenti, consultare la sezione “Creazione del file di Guida” a pagina 52-5.

Creazione del file di Guida

Per creare i file risorsa per la Guida di Windows (in formato rtf), si può utilizzare qualunque strumento. C++Builder include Microsoft Help Workshop che compila i file della Guida e mette a disposizione una Guida in linea dedicata a chi scrive tali file. Nella Guida in linea di Help Workshop si possono trovare informazioni complete sulla creazione di file di Guida.

La creazione di file di Guida per i componenti è costituita da tre fasi:

- Creazione delle voci
- Come rendere sensibile al contesto la guida del componente Aggiunta di file di Guida al componente

Creazione delle voci

Per fare in modo che la Guida del proprio componente si integri completamente con la Guida del resto dei componenti nella libreria, osservare le seguenti convenzioni:

1 **Ogni componente deve avere un argomento della guida.**

L’argomento del componente deve mostrare in quale unit il componente è dichiarato e fornirne una breve descrizione. Esso deve essere collegato ad una finestra secondaria che descrive la posizione del componente nella gerarchia degli oggetti e ne elenca tutte le proprietà, gli eventi, e i metodi. Gli sviluppatori di applicazioni accedono a questo argomento selezionando il componente in una scheda e premendo *F1*. Per un esempio di un argomento del componente, collocare un componente in una scheda e premere *F1*.

L’argomento del componente deve avere una nota a piè di pagina di tipo # con un valore univoco per identificare l’argomento. La nota a piè di pagina di tipo # serve al sistema della Guida per identificare in modo univoco ogni argomento.

L’argomento del componente deve avere una nota a piè di pagina di tipo K per la ricerca delle parole chiave nell’Indice del sistema della Guida, che includa il nome della classe del componente. Per esempio, la nota a piè di pagina per la parola chiave relativa al componente *TMemo* è “TMemo.”

Deve anche avere una nota a piè di pagina di tipo $ che fornisce il titolo all’argomento. Il titolo appare nella finestra di dialogo Topics Found, nella finestra di dialogo Bookmark e nella finestra History.

2 **Ogni componente deve includere i seguenti argomenti secondari per lo spostamento:**

- Una gerarchia di argomenti con collegamenti a ogni antenato del componente nella gerarchia dei componenti.

- Un elenco di tutte le proprietà disponibili nel componente, con collegamenti alle voci che descrivono quelle proprietà.
- Un elenco di tutti gli eventi disponibili nel componente, con collegamenti alle voci che descrivono quegli eventi.
- Un elenco di tutti i metodi disponibili nel componente, con collegamenti alle voci che descrivono quei metodi.

Utilizzando Alinks, è possibile effettuare collegamenti alle classi, alle proprietà, ai metodi o agli eventi degli oggetti nella Guida del sistema C++Builder. Quando si effettua un collegamento ad un oggetto classe, Alink utilizza il nome di classe dell'oggetto, seguito da un trattino di sottolineatura e dalla stringa "object". Per esempio, per effettuare il collegamento all'oggetto *TCustomPanel*, usare il seguente codice:

```
!AL(TCustomPanel_object,1)
```

Quando si effettua il collegamento ad una proprietà, ad un metodo o ad un evento, fare precedere il nome della proprietà, del metodo o dell'evento dal nome dell'oggetto che lo implementa e da un trattino di sottolineatura. Per esempio, per collegarsi alla proprietà *Text*, che viene implementato da *TControl*, utilizzare il seguente codice:

```
!AL(TControl_Text,1)
```

Per vedere un esempio di argomento secondario di spostamento, visualizzare la Guida per un qualsiasi componente e fare clic sui collegamenti con etichetta gerarchia, proprietà, metodi o eventi.

3 **Ogni proprietà, metodo ed evento dichiarato all'interno del componente deve avere un proprio argomento.**

Ogni argomento di una proprietà, evento o metodo deve mostrare la dichiarazione dell'elemento e descriverne l'uso. Gli sviluppatori di applicazioni possono consultare questi argomenti evidenziando l'elemento nell'Object Inspector e premendo il tasto *F1* oppure posizionando il cursore nel Code editor sul nome dell'elemento e premendo il tasto *F1*. Per vedere un esempio di un argomento di Guida relativa alle proprietà, selezionare un qualsiasi elemento nell'Object Inspector e premere *F1*.

Gli argomenti proprietà, evento e metodo devono includere una nota a piè di pagina di tipo K che elenca il nome della proprietà, del metodo o dell'evento e il suo nome in combinazione con il nome del componente. Pertanto, la proprietà *Text* di *TControl* ha la seguente nota a piè di pagina di tipo K:

```
Text,TControl;TControl,Text;Text,
```

Gli argomenti proprietà, evento e metodo devono anche includere una nota a piè di pagina di tipo $ che indica il titolo dell'argomento, come TControl::Text.

Tutti questi argomenti devono avere un ID di argomento univoco, inserito come una nota a piè di pagina di tipo #.

Come rendere sensibile al contesto la guida del componente

Ogni componente, proprietà, metodo ed evento deve avere una nota a pié di pagina di tipo A. La nota a piè di pagina A viene utilizzata per visualizzare l'argomento

quando l'utente seleziona un componente e preme *F1*, o quando una proprietà o un evento viene selezionato nell'Object Inspector e l'utente preme *F1*. La nota a piè di pagina di tipo A deve rispettare alcune convenzioni per il nome:

Se l'argomento della Guida riguarda un componente, la nota a piè di pagina di tipo A è costituita da due voci separate da un punto e virgola e utilizza questa sintassi:

```
ComponentClass_Object;ComponentClass
```

dove *ComponentClass* è il nome della classe del componente.

Se l'argomento della Guida riguarda una proprietà o un evento, la nota a piè di pagina di tipo A è costituita da tre voci separate da un punto e virgola e utilizza questa sintassi:

```
ComponentClass_Element;Element_Type;Element
```

dove *ComponentClass* è il nome della classe del componente, *Element* è il nome della proprietà, del metodo o dell'evento, e *Type* è Property, Method o Event

Per esempio, per una proprietà di nome *BackgroundColor* di un componente denominato *TMyGrid*, la nota a piè di pagina di tipo A è

```
TMyGrid_BackgroundColor;BackgroundColor_Property;BackgroundColor
```

Aggiunta di file di Guida al componente

Per aggiungere il proprio file di Guida a C++Builder, utilizzare il programma di utilità OpenHelp (chiamato oh.exe) presente nella directory bin o cui si può accedere dall'IDE tramite il comando Help | Customize.

Ulteriori informazioni sull'uso di OpenHelp sono presenti nel file OpenHelp.hlp, oltre a informazioni su come aggiungere altri file di Guida personali al sistema di Guida.

Aggiunta di editor di proprietà

L'Object Inspector fornisce funzioni predefinite di editing per tutti i tipi di proprietà. È possibile, comunque, indicare un editor alternativo per proprietà specifiche scrivendo e registrando editor di proprietà. Si possono registrare editor di proprietà validi soltanto per le proprietà dei componenti che si scrivono, ma si possono creare anche editor che si applicano a tutte le proprietà di un certo tipo.

Al livello più semplice un editor di proprietà può funzionare in uno dei seguenti metodi o in entrambi: visualizzando e permettendo all'utente di modificare il valore attuale come una stringa di testo, e visualizzando una finestra di dialogo che permette altri tipi di modifiche. In funzione della proprietà che deve essere editata, può risultare utile fornire uno o entrambi i tipi.

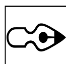
La scrittura di una proprietà editor richiede queste cinque operazioni:

1 Derivazione di una classe editor di proprietà
2 Modifica della proprietà come testo
3 Modifica globale della proprietà
4 Specifica degli attributi dell'editor

5 Registrazione dell'editor di proprietà

Derivazione di una classe editor di proprietà

Sia la CLX che la VCL definiscono diversi tipi di editor di proprietà, i quali discendono tutti da *TPropertyEditor*. Quando si crea un editor di proprietà, la classe editor di proprietà utilizzata può discendere direttamente da *TPropertyEditor* o indirettamente tramite una delle classi editor di proprietà descritte nella Tabella 52.1. Le classi nella unit DesignEditors possono essere utilizzate sia per applicazioni VCL sia per applicazioni CLX. Alcune classi editor di proprietà, tuttavia, forniscono finestre di dialogo specializzate e sono pertanto specializzate o per la VCL o per la CLX. Queste sono reperibili rispettivamente nelle unit VCLEditors e CLXEditors.



La cosa assolutamente necessaria per un editor di proprietà è che discenda da *TBasePropertyEditor* e che supporti l'interfaccia *IProperty*. *TPropertyEditor*, comunque, offre un'implementazione predefinita dell'interfaccia *IProperty*.

La lista nella Tabella 52.1 non è completa. Le unit VCLEditors e CLXEditors definiscono anche alcuni editor di proprietà specializzati, utilizzati da proprietà particolari come il nome del componente. Gli editor di proprietà elencati sono quelli più utilizzati per le proprietà definite dall'utente.

Tabella 52.1 Tipi di editor di proprietà predefiniti

Tipo	Proprietà su cui opera
TOrdinalProperty	Tutti gli editor di proprietà di tipo ordinale (cioè, proprietà di tipo integer, carattere ed enumerato) discendenti da *TOrdinalProperty*.
TIntegerProperty	Tutti i tipi integer compresi quelli predefiniti e i sottointervalli definiti dall'utente.
TCharProperty	Il tipo *Char* e tutti gli eventuali sottointervalli, come 'A'..'Z'.
TEnumProperty	Tutti i tipi enumerati.
TFloatProperty	Tutti i numeri in virgola mobile.
TStringProperty	AnsiStrings.
TSetElementProperty	I singoli elementi dei set, mostrati come valori booleani
TSetProperty	Tutti i set. I set non sono modificabili direttamente ma possono essere espansi in un elenco di proprietà elementi set.
TClassProperty	Classi. Mostra il nome della classe e permette di espanderlo per mostrare le proprietà rese pubbliche della classe.
TMethodProperty	Puntatori ai metodi, in modo particolare gli eventi.
TComponentProperty	Componenti nella stessa scheda. Non modifica le proprietà del componente ma può puntare a un componente specifico di tipo compatibile.
TColorProperty	I colori dei componenti. Mostra le costanti colore se disponibili, in caso contrario mostra un valore esadecimale. Un elenco a discesa contiene le costanti colore. Un doppio clic apre una finestra di dialogo di selezione del colore.

Tabella 52.1 Tipi di editor di proprietà predefiniti

Tipo	Proprietà su cui opera
TFontNameProperty	Nomi di font. L'elenco a discesa mostra tutti i font correntemente installati.
TFontProperty	I font. Permette l'espansione delle singole proprietà font oltre a consentire l'accesso alla finestra di dialogo dei font.

Il seguente esempio mostra la dichiarazione di un semplice editor di proprietà di nome *TMyPropertyEditor*:

```
class PACKAGE TMyPropertyEditor : public TPropertyEditor
{
  public:
    virtual bool __fastcall  AllEqual(void);
    virtual System::AnsiString __fastcall GetValue(void);
    virtual void __fastcall  SetValue(const System::AnsiString Value);
   __fastcall virtual  ~TMyPropertyEditor(void) { }
   __fastcall TMyPropertyEditor(void) : Dsgnintf::TPropertyEditor() { }
};
```

Modifica della proprietà come testo

Tutte le proprietà hanno bisogno di fornire una rappresentazione stringa dei loro valori da visualizzare nell'Object Inspector. Molte proprietà consentono, inoltre, all'utente di immettere un nuovo valore per la proprietà. Le classi editor di proprietà forniscono metodi virtuali che si possono ridefinire per effettuare la conversione tra la rappresentazione testuale e il valore effettivo.

I metodi che vengono ridefiniti si chiamano *GetValue* e *SetValue*. L'editor di proprietà eredita anche una serie di metodi utilizzati per assegnare e leggere varie specie di valori, come illustrato nella Tabella 52.2.

Tabella 52.2 Metodi per leggere e scrivere i valori delle proprietà

Tipo della proprietà	Metodo Get	Metodo Set
Numero in virgola mobile	GetFloatValue	SetFloatValue
Closure (evento)	GetMethodValue	SetMethodValue
Tipo ordinale	GetOrdValue	SetOrdValue
Stringa	GetStrValue	SetStrValue

Quando si ridefinisce un metodo *GetValue*, si chiama uno dei metodi Get; quando si ridefinisce un metodo *SetValue*, si chiama uno dei metodi Set.

Visualizzazione del valore della proprietà

Il metodo *GetValue* dell'editor di proprietà restituisce una stringa che rappresenta il valore attuale della proprietà. L'Object Inspector utilizza questa stringa nella colonna valore della proprietà. *GetValue* restituisce come valore predefinito unknown".

Per fornire alla proprietà una rappresentazione stringa, ridefinire il metodo *GetValue* dell'editor di proprietà.

Se la proprietà non è un valore stringa, *GetValue* deve convertire il valore in una rappresentazione stringa.

Impostazione del valore della proprietà

Il metodo *SetValue* dell'editor di proprietà prende una stringa scritta dall'utente nell' Object Inspector, la converte nel tipo appropriato e imposta il valore della proprietà. Se la stringa non rappresenta un valore corretto per la proprietà, *SetValue* deve generare un'eccezione e non deve utilizzare il valore improprio.

Per leggere i valori delle proprietà, ridefinire il metodo *SetValue* dell'editor di proprietà.

SetValue deve convertire la stringa e convalidare il valore prima di chiamare uno dei metodi Set.

Modifica globale della proprietà

Si può fornire opzionalmente una finestra di dialogo nella quale l'utente può editare in modo visuale una proprietà. L'impiego più comune degli editor di proprietà riguarda le proprietà che sono di per sé delle classi. Un esempio è la proprietà *Font*, per la quale l'utente può aprire una finestra di dialogo font per scegliere contemporaneamente tutti gli attributi di font.

Per fornire una finestra di dialogo per la modifica globale delle proprietà, ridefinire il metodo *Edit* della classe dell'editor di proprietà.

I metodi *Edit* usano gli stessi metodi Get e Set utilizzati nella scrittura dei metodi *GetValue* e *SetValue*. Infatti, un metodo *Edit* chiama sia un metodo Get sia un metodo Set. Poiché l'editor è specifico del tipo, generalmente non c'è bisogno di convertire i valori delle proprietà in stringhe. L'editor generalmente tratta con il valore così come lo rileva.

Quando l'utente fa clic sul pulsante '...' vicino alle proprietà o fa doppio clic sulla colonna valore, l'Object Inspector chiama il metodo *Edit* dell'editor di proprietà.

Durante la nuova implementazione del metodo *Edit*, seguire questi passi:

1 Costruire l'editor che si sta utilizzando per la proprietà.

2 Leggere il valore attuale e assegnarlo alla proprietà usando un metodo Get.

3 Quando l'utente seleziona un nuovo valore, assegnare quel valore alla proprietà usando un metodo Set.

4 Distruggere l'editor.

Specifica degli attributi dell'editor

L'editor di proprietà deve fornire tutte quelle informazioni che l' Object Inspector può utilizzare per stabilire quali strumenti mostrare a video. Per esempio, l'Object Inspector deve sapere se la proprietà contiene delle sottoproprietà o può mostrare una lista dei valori possibili.

Per specificare gli attributi dell'editor, si deve ridefinire il metodo *GetAttributes* dell'editor di proprietà.

GetAttributes è un metodo che restituisce un insieme di valori di tipo *TPropertyAttributes* che possono includere uno o tutti i seguenti valori:

Tabella 52.3 Flag attributi di un editor di proprietà

Flag	Metodo correlato	Significato del flag, se incluso
paValueList	GetValues	L'editor può restituire un elenco di valori enumerati.
paSubProperties	GetProperties	La proprietà ha sottoproprietà che è possibile visualizzare.
paDialog	Edit	L'editor può visualizzare una finestra di dialogo per modificare l'intera proprietà.
paMultiSelect	N/D	La proprietà dovrebbe essere visualizzata quando l'utente seleziona più di un componente.
paAutoUpdate	SetValue	Aggiornare il componente dopo ogni modifica invece di aspettare la conferma del valore.
paSortList	N/D	L'Object Inspector dovrebbe ordinare l'elenco dei valori.
paReadOnly	N/D	L'utente non può modificare il valore della proprietà.
paRevertable	N/D	Attiva la voce di menu Revert to Inherited del menu contestuale dell'Object Inspector. L'elemento di menu dice all'editor della proprietà di scartare il valore corrente della proprietà e di ritornare ai valori impostati in precedenza o a quelli predefiniti.
paFullWidthName	N/D	Il valore non è necessario che sia visualizzato. L'Object Inspector utilizza invece per il nome della proprietà la sua larghezza completa.
paVolatileSubProperties	GetProperties	L'Object Inspector riaggiorna i valori di tutte le sottoproprietà ogni volta che il valore della proprietà viene modificato.
paReference	GetComponent Value	Il valore è un riferimento a qualcos'altro. Se utilizzato insieme a paSubProperties l'oggetto referenziato dovrebbe essere visualizzato come sottoproprietà di questa proprietà.

Le proprietà *Color* sono più versatili di molte altre proprietà perché consentono agli utenti di selezionarle in vari modi all'interno dell'Object Inspector: scrivendone il nome, selezionandole da una lista o usando un editor personalizzato. Il metodo

GetAttributes di *TColorProperty*, pertanto, contiene diversi attributi nel valore restituito:

```
virtual __fastcall TPropertyAttributes TColorProperty::GetAttributes()
{
  return TPropertyAttributes() << paMultiSelect << paDialog << paValueList << paRevertable;
}
```

Registrazione dell'editor di proprietà

Una volta creato l'editor di proprietà, è necessario registrarlo all'interno di C++Builder. La registrazione associa un tipo di proprietà ad un determinato editor di proprietà. L'editor può venire registrato con tutte le proprietà di un certo tipo o solo con una determinata proprietà di un tipo particolare di componente.

Per registrare un editor di proprietà, chiamare la funzione *RegisterPropertyEditor*.

RegisterPropertyEditor richiede quattro parametri:

- Un puntatore all'informazione di tipo per il tipo di proprietà da editare. Specificare le informazioni di tipo in questo modo:

  ```
  __typeinfo(TMyComponent)
  ```

- Il tipo di componente al quale questo editor si applica. Se questo parametro è null, l'editor viene applicato a tutte le proprietà di un determinato tipo.
- Il nome della proprietà. Questo parametro ha significato soltanto se il parametro precedente specifica un particolare tipo di componente. In tal caso si può specificare il nome di una particolare proprietà in quel tipo di componenti al quale questo editor si applica.
- Il tipo di editor di proprietà da usare per modificare quella specifica proprietà.

Di seguito è riportato uno stralcio di una funzione che registra gli editor per i componenti standard presenti nella Component palette:

```
namespace Newcomp
{
  void __fastcall PACKAGE Register()
  {
    RegisterPropertyEditor(__typeinfo(TComponent), 0L, "", __classid(TComponentProperty));
    RegisterPropertyEditor(__typeinfo(TComponentName), __classid(TComponent), "Name",
        __classid(TComponentNameProperty));
    RegisterPropertyEditor(__typeinfo(TMenuItem), __classid(TMenu), "",
        __classid(TMenuItemProperty));
    ⋮
  }
}
```

Le tre istruzioni di questa funzione mostrano tutti i possibili usi di *RegisterPropertyEditor*:

- La prima istruzione è la più comune. Essa registra l'editor di proprietà *TComponentProperty* per tutte le proprietà di tipo *TComponent* (o per i discendenti di *TComponent* che non hanno propri editor registrati). In generale, quando si

registra un editor di proprietà, si è creato un editor per un particolare tipo e si vuole utilizzarlo per tutte le proprietà di quel tipo, così che il secondo e il terzo parametro sono, rispettivamente, **NULL** e una stringa vuota.

- La seconda istruzione è il tipo più specifico di registrazione. Essa registra un editor per una specifica proprietà in un tipo specifico di componente. In questo caso l'editor è per la proprietà *Name* (di tipo *TComponentName*) di tutti i componenti.
- La terza istruzione è più specifica della prima, ma non così limitata come la seconda. Essa registra un editor per tutte le proprietà di tipo *TMenuItem* in componenti di tipo *TMenu*.

Categorie di proprietà

Nell'IDE, l'Object Inspector consente di nascondere e visualizzare selettivamente le proprietà in base a categorie di proprietà. Le proprietà dei nuovi componenti custom possono essere adattate a questo schema registrando le proprietà in categorie. Questa operazione deve essere eseguita contemporaneamente alla registrazione del componente chiamando *RegisterPropertyInCategory* o *RegisterPropertiesInCategory*, per la registrazione della proprietà. Utilizzare *RegisterPropertyInCategory* per registrare una singola proprietà. Utilizzare *RegisterPropertiesInCategory* per registrare più proprietà con una singola chiamata alla funzione. Queste funzioni sono definite nella unit DesignIntf.

Si noti che non è obbligatorio registrare le proprietà o registrare tutte le proprietà di un componente custom nel caso se ne registrino solo alcune. Qualsiasi proprietà che non è associata esplicitamente con una categoria viene inclusa nella categoria *TMiscellaneousCategory*. Tali proprietà sono visualizzate, o nascoste, nell'Object Inspector in base alla categoria di appartenenza predefinita.

Oltre a queste due funzioni per registrare le proprietà, esiste una funzione *IsPropertyInCategory*. Questa funzione è utile per la creazione di utilità di localizzazione, in cui si deve determinare se una proprietà risulta registrata in una data categoria di proprietà.

Registrazione di una proprietà alla volta

Per registrare una proprietà alla volta e associarla con una categoria di proprietà si utilizza la funzione *RegisterPropertyInCategory*. *RegisterPropertyInCategory* ha quattro varianti caricate in sovrapposizione, ognuna delle quali fornisce una serie diversa di criteri per identificare la proprietà nel componente custom da associare con la categoria di proprietà.

La prima variante consente di identificare la proprietà in base al nome della proprietà. La riga seguente registra una proprietà correlata alla rappresentazione visuale del componente, identificando la proprietà in base al nome "AutoSize".

```
RegisterPropertyInCategory("Visual", "AutoSize");
```

La seconda variante somiglia molto alla prima, salvo che limita la categoria solo a quelle proprietà con un nome determinato che appaiono in componenti di un

determinato tipo. L'esempio seguente registra (nella categoria '*Help and Hints*') una proprietà di nome "*HelpContext*" di un componente appartenente alla classe custom *TMyButton*.

```
RegisterPropertyInCategory("Help and Hints", __classid(TMyButton), "HelpContext");
```

La terza variante identifica la proprietà utilizzandone il tipo invece del nome. L'esempio seguente registra una proprietà in base al tipo, una classe speciale di nome TArrangement.

```
RegisterPropertyInCategory('Visual', typeid(TArrangement));
```

La variante finale utilizza entrambi i tipi della proprietà e il suo nome per identificare la proprietà. L'esempio seguente registra una proprietà in base a una combinazione del tipo, *TBitmap*, e del nome, "Pattern."

```
RegisterPropertyInCategory("Visual", typeid(TBitmap), "Pattern");
```

Per un elenco delle categorie di proprietà disponibili e una breve descrizione del loro uso, consultare la sezione Specifica delle categorie di proprietà.

Registrazione contemporanea di più proprietà

La registrazione contemporanea di più proprietà e la loro associazione con una categoria di proprietà avviene utilizzando la funzione *RegisterPropertiesInCategory*. *RegisterPropertiesInCategory* ha tre varianti caricate in sovrapposizione, ognuna delle quali fornisce una serie diversa di criteri per identificare la proprietà nel componente custom da associare alle categorie di proprietà.

La prima variante consente di identificare le proprietà in base al nome o al tipo di proprietà. L'elenco viene passato come un array di costanti. Nell'esempio seguente, tutte le proprietà che hanno come nome "Text" o che appartengono a una classe di tipo *TEdit* vengono registrate nella categoria 'Localizable.'

```
RegisterPropertiesInCategory("Localizable", ARRAYOFCONST("Text", __typeinfo(TEdit)));
```

La seconda variante consente di limitare le proprietà registrate solo a quelle che appartengono a un determinato componente. L'elenco delle proprietà da registrare include solamente i nomi e non i tipi. Ad esempio, il codice seguente registra diverse proprietà di tutti i componenti nella categoria 'Help and Hints:

```
RegisterPropertyInCategory("Help and Hints", __classid(TComponent),
ARRAYOFCONST("HelpContext", "Hint", "ParentShowHint"));
```

La terza variante consente di limitare le proprietà registrate solo a quelle che hanno un determinato tipo. Come nel caso della seconda variante, l'elenco delle proprietà da registrare può includere solamente i nomi:

```
RegisterPropertiesInCategory("Localizable", __typeinfo(TStrings), ARRAYOFCONST("Lines",
"Commands"));
```

Per un elenco delle categorie di proprietà disponibili e una breve descrizione del loro uso, consultare la sezione Specifica delle categorie di proprietà.

Specifica delle categorie di proprietà

Quando si registrano le proprietà in una categoria, è possibile utilizzare come nome della categoria qualsiasi stringa si desideri. Se si utilizza una stringa che non è stata utilizzata in precedenza, l'Object Inspector genera una nuova classe di categoria di proprietà con quel nome. È possibile anche, però, registrare le proprietà in una delle categorie native. Le categorie di proprietà native sono descritte nella Tabella 52.4.:

Tabella 52.4 Categorie di proprietà

Categoria	Scopo
Action	Proprietà relative ad azioni in esecuzione; le proprietà *Enabled* e *Hint* di *TEdit* appartengono a questa categoria.
Database	Proprietà relative a operazioni di database; le proprietà *DatabaseName* e *SQL* di *TQuery* appartengono a questa categoria.
Drag, Drop, and Docking	Proprietà relative a operazioni drag-and-drop e drag-and-dock; le proprietà *DragCursor* e *DragKind* di *TImage* appartengono a questa categoria.
Help and Hints	Proprietà relative all'utilizzo della Guida in linea o dei suggerimenti; le proprietà *HelpContext* e *Hint* di *TMemo* appartengono a questa categoria.
Layout	Proprietà relative alla rappresentazione visuale di un controllo in fase di progettazione; le proprietà *Top* e *Left* di *TDBEdit* appartengono a questa categoria.
Legacy	Proprietà relative a operazioni obsolete; le proprietà *Ctl3D* e *ParentCtl3D* di *TComboBox* appartengono a questa categoria.
Linkage	Proprietà relative all'associazione o al collegamento di un componente a un altro; la proprietà *DataSet* di *TDataSource* è in questa categoria
Locale	Proprietà relative alle impostazioni locali internazionali; le proprietà *BiDiMode* e *ParentBiDiMode* di *TMainMenu* appartengono a questa categoria.
Localizable	Proprietà che potrebbe essere necessario modificare nell versioni nazionali di un'applicazione. In questa categoria rientrano diverse proprietà di tipo string (come *Caption*), in quanto sono proprietà che determinano le dimensioni e la posizione dei controlli.
Visual	Proprietà relative alla rappresentazione visuale di un controllo in fase di esecuzione; le proprietà *Align* e *Visible* di *TScrollBox* appartengono a questa categoria.
Input	Proprietà relative all'inserimento di dati (non necessariamente collegate ad operazioni su database); le proprietà *Enabled* e *ReadOnly* di *TEdit* appartengono a questa categoria.
Miscellaneous	Proprietà che non rientrano in una categoria, che non hanno bisogno di essere raggruppate in una categoria (o che non sono state registrate esplicitamente in una certa categoria); le proprietà AllowAllUp e *Name* di TSpeedButton appartengono a questa categoria.

Utilizzo della funzione IsPropertyInCategory

Un'applicazione può interrogare le proprietà registrate esistenti per determinare se una certa proprietà è già registrata in una determinata categoria. Questa capacità può

risultare particolarmente utile in programmi di utilità per la localizzazione che eseguono un controllo preliminare sull'appartenenza ad una categoria di proprietà per compiere operazioni di localizzazione. La funzione *IsPropertyInCategory* dispone di due varianti, caricate in sovrapposizione, che consentono l'uso di criteri differenti nel determinare se una proprietà appartiene ad una certa categoria.

La prima variante consente di basare il criterio di confronto su una combinazione del tipo di classe del proprio componente e del nome della proprietà. Nella riga di comando seguente, affinché *IsPropertyInCategory* restituisca **true**, la proprietà deve appartenere al discendente *TCustomEdit*, avere come nome "*Text*", e deve essere nella categoria di proprietà di nome `'Localizable'`.

```
IsItThere = IsPropertyInCategory("Localizable", __classid(TCustomEdit), "Text");
```

La seconda variante consente di basare il criterio di confronto su una combinazione del nome di classe del proprio componente e del nome della proprietà. Nella riga di comando seguente, affinché *IsPropertyInCategory* restituisca **true**, la proprietà deve appartenere al discendente *TCustomEdit*, avere come nome "*Text*", e deve essere nella categoria di proprietà di nome `'Localizable'`.

```
IsItThere = IsPropertyInCategory("Localizable", "TCustomEdit", "Text");
```

Aggiunta di editor di componenti

Gli editor di componenti determinano che cosa accade quando si fa doppio clic sul componente nel designer e aggiungono comandi al menu contestuale che appare quando si fa clic con il pulsante destro. Essi sono anche in grado di copiare il componente nella clipboard di Windows in formati custom.

Se non viene fornito nessun editor di componente ai propri componenti, C++Builder utilizza l'editor di componente predefinito. L'editor di componente predefinito è implementato dalla classe *TDefaultEditor*. *TDefaultEditor* non aggiunge nessuna nuova voce al menu contestuale del componente. Quando si fa doppio clic sul componente, *TDefaultEditor* cerca le proprietà del componente e genera (o si sposta su) il primo gestore di eventi che trova.

Per aggiungere delle voci al menu contestuale, modificare il comportamento quando si fa doppio clic sul componente, o aggiungere nuovi formati clipboard, derivare una nuova classe da *TComponentEditor* e registrarne l'uso con il proprio componente. Nei propri metodi ridefiniti si può utilizzare la proprietà *Component* di *TComponentEditor* per accedere al componente da modificare.

L'aggiunta di un editor di componente custom è una procedura composta dalle seguenti fasi:

- Aggiunta di voci al menu contestuale
- Modifica del comportamento del doppio clic
- Aggiunta di formati nella clipboard
- Registrazione dell'editor di componente

Aggiunta di voci al menu contestuale

Quando l'utente fa clic con il tasto destro del mouse sul componente, vengono chiamati i metodi *GetVerbCount* e *GetVerb* del componente editor per creare il menu contestuale. Questi metodi possono essere ridefiniti aggiungendo comandi (verbi) al menu contestuale.

L'aggiunta di elementi al menu contestuale richiede le seguenti fasi:

- Specifica delle voci di menu
- Implementazione dei comandi

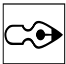
Specifica delle voci di menu

Ridefinire il metodo *GetVerbCount* per restituire il numero di comandi aggiunti al menu contestuale. Ridefinire il metodo *GetVerb* per restituire le stringhe che devono essere aggiunte ad ognuno di questi comandi. Quando si ridefinisce *GetVerb*, aggiungere una e commerciale (&) alla stringa per fare in modo che il carattere che segue la e commerciale nella stringa appaia sottolineato nel menu contestuale e funzioni come tasto di scelta rapida per la selezione della voce di menu corrispondente. Assicurarsi di aggiungere simbolo ellissi (...)alla fine di un comando se questo richiama una finestra di dialogo. *GetVerb* ha un singolo parametro che indica l'indice del comando.

Il codice seguente ridefinisce i metodi *GetVerbCount* e *GetVerb* per aggiungere due comandi al menu contestuale.

```
int __fastcall TMyEditor::GetVerbCount(void)
{
  return 2;
}

System::AnsiString __fastcall TMyEditor::GetVerb(int Index)
{
  switch (Index)
  {
    case 0: return "&DoThis ..."; break;
    case 1: return "Do&That"; break;
  }
}
```



È bene assicurarsi che il proprio metodo *GetVerb* restituisca un valore per ogni possibile indice indicato da *GetVerbCount*.

Implementazione dei comandi

Quando si seleziona un comando fornito da *GetVerb* nella funzione di impostazione, viene chiamato il metodo *ExecuteVerb*. Per ogni comando fornito nel metodo *GetVerb*, implementare un'azione nel metodo *ExecuteVerb*. Utilizzando la proprietà *Component* dell'editor, si può accedere al componente da modificare.

Ad esempio, il seguente metodo *ExecuteVerb* implementa i comandi per il metodo *GetVerb* dell'esempio precedente.

```
void __fastcall TMyEditor::ExecuteVerb(int Index)
{
```

```
    switch (Index)
    {
      case 0:
        TMyDialog *MySpecialDialog = new TMyDialog();
        MySpecialDialog->Execute();
        ((TMyComponent *)Component)->ThisProperty = MySpecialDialog->ReturnValue;
        delete MySpecialDialog;
        break;
      case 1:
        That();  // call the "That" method
        break;
    }
  }
```

Modifica del comportamento del doppio clic

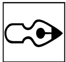
Quando si fa doppio clic sul componente, viene chiamato il metodo *Edit* dell'editor di componente. Il metodo *Edit*, come impostazione predefinita, esegue il primo comando aggiunto al menu contestuale. Quindi, nell'esempio precedente, facendo doppio clic sul componente viene eseguito il comando *DoThis*.

Anche se l'esecuzione del primo comando è generalmente una buona idea, si può anche decidere di cambiare questo comportamento predefinito. Ad esempio, si può voler fornire un comportamento alternativo se

- non si aggiunge nessun comando al menu contestuale.
- si vuole che compaia una finestra di dialogo che combina diversi comandi quando si fa doppio clic sul componente.

Ridefinire il metodo *Edit* per specificare un nuovo comportamento quando si fa doppio clic sul componente. Per esempio, il seguente metodo *Edit* apre una finestra di dialogo font quando l'utente fa doppio clic sul componente:

```
void __fastcall TMyEditor::Edit(void)
{
  TFontDialog *pFontDlg = new TFontDialog(NULL);
  pFontDlg->Execute();
  ((TMyComponent *)Component)->Font = pFontDlg->Font;
  delete pFontDlg;
}
```



Se si vuole visualizzare il Code editor per un gestore di eventi quando si fa doppio clic sul componente, utilizzare *TDefaultEditor* come classe basse per il proprio editor del componente al posto di *TComponentEditor*. Quindi, invece di ridefinire il metodo *Edit*, ridefinire in sua vece il metodo protetto *TDefaultEditor*::*EditProperty*. *EditProperty* effettua un scansione dei gestori di eventi del componente, e attiva il primo che trova. Questo processo può essere modificato facendo sì che venga cercato uno specifico gestore di eventi. Per esempio:

```
void __fastcall TMyEditor::EditProperty(TPropertyEditor* PropertyEditor,
                                        bool &Continue, bool &FreeEditor)
{
  if (PropertyEditor->ClassNameIs("TMethodProperty") &&
```

```
        CompareText(PropertyEditor->GetName, "OnSpecialEvent") == 0)
    {
      TDefaultEditor::EditProperty(PropertyEditor, Continue, FreeEditor);
    }
  }
```

Aggiunta di formati nella clipboard

Per impostazione predefinita, quando un utente sceglie Copy mentre c'è un componente selezionato nell'IDE, il componente viene copiato utilizzando il formato interno di C++Builder. Tale componente può, a questo punto, essere incollato in un'altra scheda o in un modulo dati. Un componente può copiare ulteriori formati nella clipboard di Windows ridefinendo il metodo *Copy*.

Per esempio, il seguente metodo *Copy* permette a un componente *TImage* di copiare la sua immagini nella clipboard. Questa immagine viene ignorata dall'IDE di C++Builder, ma può essere incollata in un'altra applicazione.

```
void __fastcall TMyComponentEditor::Copy(void)
{
  WORD AFormat;
  int AData;
  HPALETTE APalette;
  ((TImage *)Component)->Picture->SaveToClipboardFormat(AFormat, AData, APalette);
  TClipboard *pClip = Clipboard(); // don't clear the clipboard!
  pClip->SetAsHandle(AFormat, AData);
}
```

Registrazione dell'editor di componente

Una volta definito l'editor del componente, può essere registrato in modo che sia in grado di operare con una particolare classe di componenti. Viene creato un editor di componente registrato per ogni componente di quella classe quando lo si seleziona nel Form designer.

Per creare l'associazione tra un editor di componente e una classe di componente, chiamare *RegisterComponentEditor*. *RegisterComponentEditor* Prende il nome della classe del componente che utilizza l'editor, e il nome della classe dell'editor del componente che è stato definito. Per esempio, la seguente istruzione registra una classe dell'editor di componente chiamata *TMyEditor* perché funzioni con tutti i componenti di tipo *TMyComponent*:

```
RegisterComponentEditor(__classid( TMyComponent), __classid(TMyEditor));
```

Collocare la chiamata a *RegisterComponentEditor* nel namespace in cui si registra il componente. Per esempio, se un nuovo componente di nome *TMyComponent* e il suo editor di componente *TMyEditor* sono entrambi implementati in NewComp.cpp, il codice seguente (in NewComp.cpp) registra il componente e la sua associazione con il componente editor.

```
namespace Newcomp
{
```

```
    void __fastcall PACKAGE Register()
    {
      TMetaClass classes[1] = {__classid(TMyComponent)};
      RegisterComponents("Miscellaneous", classes, 0);
      RegisterComponentEditor(classes[0], __classid(TMyEditor));
    }
  }
```

Compilazione dei componenti in package

Una volta che i componenti sono stati registrati, occorre compilarli come package prima di poterli installare nell'IDE. Un package può contenere uno o più componenti oltre a vari editor di proprietà personalizzati. Per ulteriori informazioni sui package, consultare ilCapitolo 15, "Uso di package e di componenti".

Per creare e compilare un package, seguire le istruzioni contenute nel paragrafo "Creazione ed editing dei package" a pagina 15-7. Inserire le unit del codice sorgente per i propri componenti personalizzati nella lista Contains del package. Se i propri componenti dipendono da altri package, includere tali package nella lista Requires.

Per installare i propri componenti nell'IDE, seguire le istruzioni riportate nel paragrafo "Installazione di package di componenti" a pagina 15-6.

Risoluzione dei problemi con componenti custom

Un problema frequente, che si verifica durante la registrazione e l'installazione di componenti custom, è che il componente non viene visualizzato nella lista dei componenti, pur avendo installato con successo il package.

Le cause più comuni che impediscono la visualizzazione dei componenti nell'elenco o nella tavolozza sono:

- Mancanza del modificatore PACKAGE nella funzione Register
- Mancanza del modificatore PACKAGE nella classe
- Mancanza della direttiva `#pragma package(smart_init)` nel file sorgente in C++
- La funzione Register non è presente in un namespace avente lo stesso nome del nome del modulo del codice sorgente.
- La funzione Register non viene esportata con successo. Utilizzare il programma di utilità TDUMP sul file .BPL per esaminare le funzioni esportate:

  ```
  tdump -ebpl mypack.bpl  mypack.dmp
  ```

  Nella sezione del dump relativa all'esportazione, si dovrebbe vedere che la funzione Register (all'interno del namespace) viene esportata.

Capitolo

53

Modifica di un componente esistente

Il modo più semplice per creare un componente consiste nel derivarlo da un altro quanto più possibile simile a quello desiderato, apportandovi poi le modifiche necessarie. L'esempio riportato in questo capitolo modifica il componente standard memo per creare un memo con lo sfondo di colore giallo. Negli altri capitoli di questa sezione viene descritta la creazione di componenti più complessi. Il processo base è sempre lo stesso, ma i componenti più complessi richiedono un maggior numero di operazioni per personalizzare la nuova classe.

La modifica di un componente esistente richiede solamente due fasi:

- Creazione e registrazione del componente
- Modifica della classe del componente

Creazione e registrazione del componente

La creazione di qualunque componente comincia nello stesso modo: si deriva la classe di un componente, si salvano i file .CPP e .H del componente, si registra il componente, lo si compila e lo si installa nella Component palette. Questo processo viene descritto nella sezione "Creazione di un nuovo componente" a pagina 45-8.

Per creare un componente con le specifiche desiderate, seguire questa procedura:

1. Assegnare alla unit del componente il nome *YelMemo* e salvarla, in modo che il file header sia YELMEMO.H e il file .CPP sia YELMEMO.CPP.
2. Derivare un nuova classe di componente di nome *TYellowMemo*, facendolo discendere dalla classe *TMemo*.
3. Registrare *TYellowMemo* nella pagina Samples (o in un'altra pagina della CLX) della Component palette.

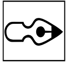
Il file header risultante dovrebbe apparire simile al seguente:

```
#ifndef YelMemoH
#define YelmemoH
//---------------------------------------------------------------------------
#include <sysutils.hpp>
#include <controls.hpp>
#include <classes.hpp>
#include <forms.hpp>
#include <StdCtrls.hpp>
//---------------------------------------------------------------------------
class PACKAGE TYellowMemo : public TMemo
{
private:
protected:
public:
__published:
};
//---------------------------------------------------------------------------
#endif
```

Il file .CPP associato sarà più o meno come questo:

```
#include <vcl.h>
#pragma hdrstop
#include "Yelmemo.h"
//---------------------------------------------------------------------------
#pragma package(smart_init);
//---------------------------------------------------------------------------
// ValidCtrCheck is used to assure that the components created do not have
// any pure virtual functions.
//
static inline void ValidCtrCheck(TYellowMemo *)
{
  new TYellowMemo(NULL);
}
//---------------------------------------------------------------------------
__fastcall TYellowMemo::TYellowMemo(TComponent* Owner)
        : TMemo(Owner)
{
}
//---------------------------------------------------------------------------
namespace Yelmemo
{
  void __fastcall PACKAGE Register()
  {
    TComponentClass classes[1] = {__classid(TYellowMemo)};
    RegisterComponents("Samples", classes, 0); //"Common Controls" in CLX applications
  }
}
```

Nelle applicazioni CLX i nomi e le ubicazioni di alcuni file header sono differenti. Ad esempio, `<vcl\controls.hpp>` è `<clx\qcontrols.hpp>` in CLX.



In questo esempio si presuppone che non si utilizzi il Component wizard per creare questo componente, ma che si proceda manualmente. Se si usa il Component wizard, verrà aggiunto automaticamente un costruttore a *TYellowMemo*.

Modifica della classe del componente

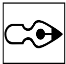
Una volta creata una nuova classe di componente, è possibile modificarla praticamente in qualsiasi modo. In questo caso, si modificherà solamente il valore iniziale di una proprietà del componente memo. Ciò comporta due piccole modifiche alla classe del componente:

- Ridefinizione del costruttore.
- Specifica del nuovo valore predefinito di una proprietà.

È il costruttore che, di fatto, imposta il valore della proprietà. Il valore predefinito impostato dice a C++Builder quali valori memorizzare nel file scheda (.dfm per la VCL e .xfm per la CLX). C++Builder memorizza solamente i valori che differiscono da quelli predefiniti, quindi è importante compiere entrambe queste operazioni.

Ridefinizione del costruttore

Quando un componente viene collocato su una scheda in fase di progetto, o quando un'applicazione costruisce un componente durante l'esecuzione, il costruttore del componente imposta i valori delle relative proprietà. Quando un componente viene caricato da un file scheda, l'applicazione imposta tutte le proprietà modificate in fase di progetto.

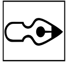


Quando si ridefinisce un costruttore, il nuovo costruttore deve chiamare il costruttore ereditato prima di fare qualsiasi altra cosa. Per ulteriori informazioni consultare il paragrafo "Ridefinizione dei metodi" a pagina 46-10.

In questo esempio, il nuovo componente necessita della ridefinizione del costruttore ereditato da *TMemo* per impostare la proprietà *Color* a *clYellow*. A questo scopo, aggiunge la dichiarazione di ridefinizione del costruttore alla dichiarazione della classe, quindi scrive il nuovo costruttore nel file .CPP:

```
class PACKAGE TYellowMemo : public TMemo
{
public:
    virtual __fastcall TYellowMemo(TComponent* Owner); // the constructor declaration
__published:
    __property Color;
};

__fastcall TYellowMemo::TYellowMemo(TComponent* Owner)
  : TMemo(Owner)                                   // the constructor implementation first...
                                                    // ...calls the constructor for TMemo
{
  Color = clYellow;                               // colors the component yellow
}
```



Se per creare il componente si è utilizzato il Component wizard, è sufficiente soltanto aggiungere `Color = clYellow;` al costruttore esistente.

A questo punto è possibile installare il nuovo componente nella Component palette e aggiungerlo alla scheda. La proprietà *Color* predefinita ora è *clYellow*.

Specifica del nuovo valore predefinito di una proprietà

Quando C++Builder memorizza la descrizione di una scheda in un file scheda, memorizza solo i valori delle proprietà che differiscono da quelli predefiniti. La memorizzazione dei soli valori che differiscono da quelli predefiniti aiuta a mantenere piccoli i file di scheda e rende più rapido il caricamento della scheda. Se si crea una proprietà o si modifica il valore predefinito, può essere utile aggiornare la dichiarazione della proprietà per includere il nuovo valore predefinito. I file di scheda, il caricamento e i valori predefiniti sono spiegati più dettagliatamente nel Capitolo 52, “Come rendere disponibili i componenti in fase di progetto”.

Per modificare il valore predefinito di una proprietà:

1 Ridichiarare il nome della proprietà.

2 Porre un segno uguale (=) dopo il nome della proprietà.

3 Scrivere la parola chiave **default**, un altro segno uguale e il valore predefinito, racchiudendo il tutto fra parentesi graffe.

Si noti che non è necessario dichiarare di nuovo l’intera proprietà, ma solo il nome e il valore predefinito.

Per il componente memo yellow, si deve dichiarare di nuovo la proprietà *Color* in una sezione **__published** della dichiarazione della classe, con un valore predefinito pari a *clYellow*:

```
class PACKAGE TYellowMemo : public TMemo
{
public:
    virtual __fastcall TYellowMemo(TComponent* Owner);
__published:
    __property Color = {default=clYellow};
};
```

Ecco di nuovo *TYellowMemo*, ma questa volta ha un’altra proprietà published, *WordWrap*, con un valore predefinito impostato a **false**:

```
class PACKAGE TYellowMemo : public TMemo
{
public:
    virtual __fastcall TYellowMemo(TComponent* Owner);
__published:
    __property Color = {default=clYellow};
    __property WordWrap = {default=false};
};
```

La specifica del valore predefinito della proprietà non influenza minimamente il funzionamento del componente. Occorre ancora impostare esplicitamente il valore predefinito del costruttore del componente. La differenza sta nel funzionamento interno dell'applicazione. Per *TYellowMemo*, C++Builder non scriverà più *WordWrap* nel file scheda se è **false**, perché gli è stato comunicato che il costruttore imposterà automaticamente quel valore. Ecco il codice del costruttore:

```
__fastcall TYellowMemo::TYellowMemo(TComponent* AOwner) : TMemo(AOwner)
{
  Color = clYellow;
  WordWrap = false;
}
```

Capitolo

54

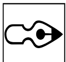
Creazione di un controllo grafico

Un controllo grafico è un tipo di componente semplice. Poiché un controllo esclusivamente grafico non riceve mai il fuoco, non possiede né richiede un handle di finestra. Gli utenti possono comunque gestire il controllo col mouse, ma non c'è una interfaccia di tastiera.

Il componente grafico presentato in questo capitolo è *TShape*, il componente shape che appare nella pagina Additional della Component palette. Anche se il componente che viene creato è identico al componente shape standard, è necessario chiamarlo in modo diverso per evitare duplicati di identificatori. In questo capitolo al componente viene assegnato il nome *TSampleShape* e vengono illustrati tutti i passi necessari alla sua creazione:

Creazione e registrazione del componente

La creazione di qualunque componente inizia nello stesso modo: si deriva una classe di componente, si salvano i file .CPP e .H del componente, si registra il componente, lo si compila e lo si installa nella Component palette. Questo processo viene descritto nella sezione "Creazione di un nuovo componente" a pagina 45-8.



In questo esempio si presuppone che non si utilizzi Component wizard, ma che si crei il componente manualmente.

Per questo esempio, seguire la procedura generale di creazione di un componente, con queste specifiche:

1 Derivare un nuovo tipo di componente chiamato *TSampleShape*, discendente da *TGraphicControl*.

2 Chiamare il file header del componente SHAPES.H e il file .CPP associato SHAPES.CPP.

3 Registrare *TSampleShape* nella pagina Samples (o in un'altra pagina della CLX) della Component palette.

Il file header risultante sarà simile al seguente:

```
//---------------------------------------------------------------------------
#ifndef ShapesH
#define ShapesH
//---------------------------------------------------------------------------
#include <sysutils.hpp>
#include <controls.hpp>
#include <classes.hpp>
#include <forms.hpp>
//---------------------------------------------------------------------------
class PACKAGE TSampleShape : public TGraphicControl
{
private:
protected:
public:
__published:
};
//---------------------------------------------------------------------------
#endif
```

Il file .CPP sarà più o meno come questo:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Shapes.h"
//---------------------------------------------------------------------------
#pragma package(smart_init);
//---------------------------------------------------------------------------
// ValidCtrCheck is used to assure that the components created do not have
// any pure virtual functions.
//

static inline void ValidCtrCheck(TSampleShape *)
{
  new TSampleShape(NULL);
}
//---------------------------------------------------------------------------
__fastcall TSampleShape::TGraphicControl(TComponent* Owner)
        : TGraphicControl(Owner)
{
}
//---------------------------------------------------------------------------
namespace Shapes
{
  void __fastcall PACKAGE Register()
  {
    TComponentClass classes[1] = {__classid(TSampleShape)};
    RegisterComponents("Samples", classes, 0);
```

```
    }
}
```

Nelle applicazioni CLX i nomi e le ubicazioni di alcuni file header sono differenti. Ad esempio, <vcl\controls.hpp> è <clx\qcontrols.hpp> in CLX.

Pubblicazione delle proprietà ereditate

Una volta derivato un tipo di componente, si può decidere quali proprietà ed eventi, dichiarati nelle sezioni protette della classe antenato, si vogliono rendere disponibili nel nuovo componente. *TGraphicControl* pubblica già tutte le proprietà che consentono al componente di funzionare come un controllo, per cui, l'unica cosa che occorre rendere pubblica è la capacità di rispondere agli eventi del mouse e di gestire le operazioni drag-and-drop.

Il modo per rendere pubbliche le proprietà e gli eventi ereditati è spiegato nel paragrafo "Pubblicazione di proprietà ereditate" a pagina 47-3 e "Visibilità degli eventi" a pagina 48-5. Entrambi i processi includono la ridichiarazione del nome delle proprietà nella sezione **published** della dichiarazione della classe.

Per il controllo shape è possibile rendere pubblici i tre eventi del mouse, le due proprietà e i tre eventi relativi alle operazioni di drag-and-drop:

```
class PACKAGE TSampleShape : public TGraphicControl
{
private:
__published:
    __property DragCursor ;
    __property DragMode ;
    __property OnDragDrop ;
    __property OnDragOver ;
    __property OnEndDrag ;
    __property OnMouseDown ;
    __property OnMouseMove ;
    __property OnMouseUp ;
};
```

Il controllo shape dell'esempio rende disponibili agli utenti le interazioni del mouse e il drag-and-drop.

Aggiunta di funzionalità grafiche

Una volta dichiarato il componente grafico e resa pubblica ogni proprietà ereditata che si vuole rendere disponibile, è possibile aggiungere alcune funzionalità grafiche che distinguano il componente. Durante la creazione di un controllo grafico, vengono compiute due operazioni:

1. Determinazione di cosa disegnare.
2. Disegno dell'immagine del componente.

Inoltre, al nostro controllo shape di esempio verranno aggiunte alcune proprietà che permetteranno agli sviluppatori di applicazioni di personalizzare il controllo modificandone l'aspetto durante lo sviluppo.

Determinazione di cosa disegnare

Un controllo grafico può modificare il proprio aspetto per riflettere una condizione dinamica, incluso l'input dell'utente. Un controllo grafico che non varia mai non sarebbe probabilmente neanche un componente. Se si vuole un'immagine statica, invece di usare un controllo, la si può importare.

In generale, l'aspetto di un controllo grafico dipende da alcune combinazioni delle proprietà. Il controllo Gauge, per esempio, ha proprietà che ne determinano forma e orientamento oltre a determinare se mostrare l'evolversi di un processo numericamente e/o graficamente. Similmente, il controllo shape ha proprietà che determinano quale tipo di figura deve tracciare.

Per attribuire al controllo una proprietà che determini la figura da disegnare, aggiungere un proprietà di nome *Shape*.Ciò richiede

1. Dichiarzione del tipo della proprietà.
2. Dichiarazione della proprietà.
3. Scrittura del metodo di implementazione.

La creazione di proprietà è illustrata più dettagliatamente nel Capitolo 47, "Creazione di proprietà".

Dichiarzione del tipo della proprietà

Quando si dichiara la proprietà di un tipo definito dall'utente, prima della classe che include la proprietà si deve dichiarare il tipo. La categoria più comune dei tipi di proprietà definiti dall'utente è quello enumerato.

Aggiungere la seguente definizione di tipo prima della dichiarazione della classe del controllo shape.

```
enum TSampleShapeType { sstRectangle, sstSquare, sstRoundRect, sstRoundSquare, sstEllipse,
sstCircle };

class PACKAGE TSampleShape : public TGraphicControl        // this is already there
```

È possibile ora utilizzare questo tipo per dichiarare una nuova proprietà all'interno della classe.

Dichiarazione della proprietà

Quando si dichiara una proprietà, solitamente è necessario dichiarare un membro dati privato per registrare i dati delle proprietà, quindi specificare i metodi di lettura e/o scrittura del valore della proprietà. Spesso non è necessario utilizzare un metodo per leggere il valore, ma è sufficiente puntare ai dati registrati.

Per il controllo shape bisogna dichiarare un membro dati che contenga la figura attiva, quindi dichiarare una proprietà che legga e scriva tale membro dati richiamando un metodo.

Aggiungere la seguente dichiarazione a *TSampleShape*:

```
class PACKAGE TSampleShape : public TGraphicControl
{
private:
    TSampleShapeType FShape;
    void __fastcall SetShape(TSampleShapeType Value);
__published:
    __property TSampleShapeType Shape = {read=FShape, write=SetShape, nodefault};
};
```

Ora tutto ciò che rimane da fare è aggiungere l'implementazione di *SetShape*.

Scrittura del metodo di implementazione

Quando la parte **read** o **write** della definizione della proprietà usa un metodo invece di accedere direttamente ai dati memorizzati della proprietà, occorre implementare il metodo.

Aggiungere l'implementazione del metodo *SetShape* al file SHAPES.CPP:

```
void __fastcall TSampleShape::SetShape(TSampleShapeType Value)
{
  if (FShape != Value)              // ignore if this isn't a change
  {
     FShape = Value;                // store the new value
     Invalidate();                  // force a repaint with the new shape
  }
}
```

Ridefinizione del costruttore e del distruttore

Per modificare i valori predefiniti della proprietà e inizializzare le classi possedute dal componente, si deve ridefinire il costruttore e il distruttore ereditati. In entrambi i casi, bisogna ricordarsi di chiamare il metodo ereditato nel nuovo costruttore o distruttore.

Cambiamento dei valori predefiniti delle proprietà

Poiché la dimensione predefinita di un controllo grafico è piuttosto piccola, è possibile cambiarne la larghezza e l'altezza all'interno del metodo costruttore. Il cambiamento dei valori predefiniti delle proprietà è spiegato in maniera più dettagliata nel Capitolo 53, "Modifica di un componente esistente".

In questo esempio il controllo shape imposta le proprie dimensioni ad un quadrato di 65 pixel per lato.

Aggiungere il costruttore ridefinito alla dichiarazione della classe del componente:

```
class PACKAGE TSampleShape : public TGraphicControl
{
public:
    virtual __fastcall TSampleShape(TComponent *Owner);
};
```

Se si è iniziata la creazione del componente mediante il wizard Component, questa operazione è già stata eseguita automaticamente.

1 Ridichiarare le proprietà *Height* e *Width* con i loro nuovi valori predefiniti:

```
class PACKAGE TSampleShape : public TGraphicControl
{
  ⋮
__published:
    __property Height;
    __property Width;
}
```

2 Scrivere il nuovo costruttore nel file .CPP:

```
__fastcall TSampleShape::TSampleShape(TComponent* Owner) : TGraphicControl(Owner)
{
  Width = 65;
  Height = 65;
}
```

Se si utilizza il wizard Component, tutto ciò che occorre fare è aggiungere i nuovi valori predefiniti al costruttore già esistente.

Pubblicazione di penna e pennello

Per impostazione predefinita, un canvas ha una penna nera e sottile e un pennello bianco e pieno. Per consentire agli sviluppatori di modificare la penna e il pennello, si devono preparare le relative classi da gestire in fase di progetto e successivamente copiarle nel canvas durante la creazione delle immagini. Le classi, quali ad esempio, una penna o un pennello aggiuntivi, sono chiamate classi possedute perché il componente le possiede ed è responsabile della loro creazione e distruzione.

Per gestire le classi possedute occorre seguire le seguenti operazioni:

1 Dichiarazione dei membri dati della classe.
2 Dichiarazione delle proprietà di accesso.
3 Inizializzazione delle classi possedute.
4 Impostazione delle proprietà delle classi possedute.

Dichiarazione dei membri dati della classe

Ciascuna classe posseduta da un componente deve avere un proprio membro dati dichiarato nel componente. Il membro dati assicura che il componente abbia sempre un puntatore all'oggetto posseduto così da poter distruggere la classe prima di distruggere se stesso. In generale, un componente inizializza gli oggetti posseduti nel proprio costruttore e li distrugge nel proprio distruttore.

I membri dati degli oggetti posseduti sono quasi sempre dichiarati come privati. Se le applicazioni (o gli altri componenti) devono accedere agli oggetti posseduti, si possono dichiarare appositamente proprietà **published** o **public**.

Aggiungere i membri dati per una penna ed un pennello al controllo shape:

```
class PACKAGE TSampleShape : public TGraphicControl
{
private:              // data members are always private
    TPen *FPen;       // a data member for the pen object
    TBrush *FBrush;   // a data member for the brush object
    ⋮
};
```

Dichiarazione delle proprietà di accesso

È possibile garantire l’accesso agli oggetti posseduti di un componente dichiarando le proprietà del tipo degli oggetti.Questo dà modo agli sviluppatori di accedere agli oggetti in fase di progetto o durante l’esecuzione. Solitamente, la parte **read** della proprietà referenzia solo il membro dati, ma la parte **write** chiama un metodo che consente al componente di reagire alle modifiche nell’oggetto posseduto.

Aggiungere al controllo shape le proprietà che forniscono l’accesso ai membri dati penna e pennello. Si dichiareranno anche i metodi per reagire alle modifiche apportate alla penna o al pennello.

```
class PACKAGE TSampleShape : public TGraphicControl
{
  ⋮
private:
    TPen *FPen;
    TBrush *FBrush;
    void __fastcall SetBrush(TBrush *Value);
    void __fastcall SetPen(TPen *Value);
    ⋮
__published:
    __property TBrush* Brush = {read=FBrush, write=SetBrush, nodefault};
    __property TPen* Pen = {read=FPen, write=SetPen, nodefault};
};
```

Quindi, scrivere i metodi *SetBrush* e *SetPen* nel file .CPP:

```
void __fastcall TSampleShape::SetBrush( TBrush* Value)
{
  FBrush->Assign(Value);
}

void __fastcall TSampleShape::SetPen( TPen* Value)
{
  FPen->Assign(Value);
}
```

L’assegnazione diretta del contenuto di *Value* a *FBrush* nel seguente modo

```
FBrush = Value;
```

avrebbe sovrascritto il puntatore interno per *FBrush* e azzerato la memoria, creando un buon numero di problemi di proprietà.

Inizializzazione delle classi possedute

Se si aggiungono delle classi al proprio componente, il costruttore del componente deve inizializzarle in modo che l’utente finale possa interagire con gli oggetti in fase

di esecuzione. In modo analogo, il distruttore del componente deve distruggere gli oggetti posseduti prima di distruggere se stesso.

Poiché al controllo shape sono stati aggiunti una penna e un pennello, è necessario inizializzarli nel costruttore del controllo shape e distruggerli nel distruttore del controllo:

1 Costruire la penna e il pennello nel costruttore del controllo shape:

```
__fastcall TSampleShape::TSampleShape(TComponent* Owner) : TGraphicControl(Owner)
{
  Width = 65;
  Height = 65;
  FBrush = new TBrush();                           // construct the pen
  FPen = new TPen();                               // construct the brush
}
```

2 Aggiungere il distruttore ridefinito alla dichiarazione della classe del componente:

```
class PACKAGE TSampleShape : public TGraphicControl
{
  ⋮
public:                                            // destructors are always public
    virtual __fastcall TSampleShape(TComponent* Owner);
    __fastcall ~TSampleShape();                    // the destructor
    ⋮
};
```

3 Scrivere il nuovo distruttore nel file .CPP:

```
__fastcall TSampleShape::~TSampleShape()
{
  delete FPen;                                     // delete the pen object
  delete FBrush;                                   // delete the brush object
}
```

Impostazione delle proprietà delle classi possedute

Come ultimo passo della gestione delle classi penna e pennello, bisogna accertarsi che i cambiamenti nella penna e nel pennello forzino il controllo shape a ridisegnarsi. La penna e il pennello hanno entrambi eventi *OnChange*, quindi si può creare un metodo nel controllo shape e far puntare ad esso entrambi gli eventi *OnChange*.

Aggiungere il metodo seguente al controllo shape e aggiornare il costruttore del componente per assegnare gli eventi di penna e pennello al nuovo metodo:

```
class PACKAGE TSampleShape : public TGraphicControl
{
  ⋮
public:
    void __fastcall StyleChanged(TObject* Owner);
    ⋮
};
```

Assegnare il metodo *StyleChanged* agli eventi *OnChange* delle classi penna e pennello nel costruttore *TSampleShape* nel file SHAPES.CPP:

```
__fastcall TSampleShape::TSampleShape(TComponent* Owner) : TGraphicControl(Owner)
{
```

```
    Width = 65;
    Height = 65;
    FBrush = new TBrush();
    FBrush->OnChange = StyleChanged;
    FPen = new TPen();
    FPen->OnChange = StyleChanged;
  }
```

Includere l'implementazione del metodo *StyleChanged*:

```
void __fastcall TSampleShape::StyleChanged( TObject* Sender)
{
  Invalidate();         // repaints the component
}
```

Dopo queste modifiche il componente viene ridisegnato in modo da rispecchiare i cambiamenti della penna e del pennello.

Disegno dell'immagine del componente

L'elemento più importante di un controllo grafico è il metodo con cui disegna la propria immagine sullo schermo. Il tipo astratto *TGraphicControl* definisce un metodo di nome *Paint* che è possibile ridefinire per disegnare l'immagine prescelta per il controllo.

Il metodo *Paint* per il controllo shape necessita di:

- Utilizzare la penna e il pennello selezionati dall'utente.
- Utilizzare la forma selezionata.
- Adeguare le coordinate in modo che quadrati e cerchi usino la stessa larghezza e la stessa altezza.

La ridefinizione del metodo *Paint* implica due operazioni:

1. Aggiungere *Paint* alla dichiarazione del componente.
2. Scrivere il metodo *Paint* nel file .CPP.

Per il controllo shape aggiungere la seguente dichiarazione alla dichiarazione della classe:

```
class PACKAGE TSampleShape : public TGraphicControl
{
  ⋮
protected:
    virtual void __fastcall Paint();
    ⋮
};
```

Quindi, scrivere il metodo nel file. CPP:

```
void __fastcall TSampleShape::Paint()
{
  int X,Y,W,H,S;
  Canvas->Pen = FPen;                            // copy the component's pen
  Canvas->Brush = FBrush;                        // copy the component's brush
  W=Width;                                       // use the component width
```

```
    H=Height;                                      // use the component height
    X=Y=0;                                         // save smallest for circles/squares
    if( W<H )
      S=W;
    else
      S=H;
    switch(FShape)
    {
      case sstRectangle:                           // draw rectangles and squares
      case sstSquare:
        Canvas->Rectangle(X,Y,X+W,Y+H);
        break;
      case sstRoundRect:                           // draw rounded rectangles and squares
      case sstRoundSquare:
        Canvas->RoundRect(X,Y,X+W,Y+H,S/4,S/4);
        break;
      case sstCircle:                              // draw circles and ellipses
      case sstEllipse:
        Canvas->Ellipse(X,Y,X+W,Y+H);
        break;
      default:
        break;
    }
  }
```

Paint viene chiamato ogni volta che il controllo ha bisogno di aggiornare la sua immagine. I controlli vengono disegnati quando essi appaiono per la prima volta o quando scompare una finestra di fronte ad loro. È anche possibile forzare la rigenerazione dell'immagine chiamando *Invalidate*, come fa il metodo *StyleChanged*.

Perfezionamento del disegno della figura

Il controllo shape standard fa una cosa in più rispetto al controllo shape di esempio: gestisce non solo i quadrati e i cerchi, ma anche i rettangoli e le ellissi. Per fare ciò, occorre scrivere del codice che in grado di individuare il lato più corto e di centrare l'immagine.

Qui di seguito viene riportato il metodo *Paint* perfezionato che corregge i quadrati e le ellissi:

```
void __fastcall TSampleShape::Paint(void)
{
  int X,Y,W,H,S;
  Canvas->Pen = FPen;                     // copy the component's pen
  Canvas->Brush = FBrush;                 // copy the component's brush
  W=Width;                                // use the component width
  H=Height;                               // use the component height
  X=Y=0;                                  // save smallest for circles/squares
  if( W<H )
    S=W;
  else
    S=H;
  switch(FShape)                          // adjust height, width and position
```

```
  {
    case sstRectangle:
    case sstRoundRect:
    case sstEllipse:
      Y=X=0;                          // origin is top-left for these shapes
      break;
    case sstSquare:
    case sstRoundSquare:
    case sstCircle:
      X= (W-S)/2;                     // center these horizontally
      Y= (H-S)/2;                     // and vertically
      break;
    default:
    break;
  }
  switch(FShape)
  {
    case sstSquare:                   // draw rectangles and squares
      W=H=S;                          // use shortest dimension for width and height
    case sstRectangle:
      Canvas->Rectangle(X,Y,X+W,Y+H);
      break;
    case sstRoundSquare:              // draw rounded rectangles and squares
      W=H=S;
    case sstRoundRect:
      Canvas->RoundRect(X,Y,X+W,Y+H,S/4,S/4);
      break;
    case sstCircle:                   // draw circles and ellipses
      W=H=S;
    case sstEllipse:
      Canvas->Ellipse(X,Y,X+W,Y+H);
      break;
    default:
      break;
  }
}
```

Capitolo

55

Personalizzazione di una griglia

C++Builder è dotato di componenti astratti utilizzabili come basi per i componenti personalizzati. I più importanti sono le griglie e le liste. In questo capitolo è possibile vedere come si crea un piccolo calendario mensile partendo dalla griglia base *TCustomGrid*.

La creazione del calendario richiede le seguenti operazioni:

- Creazione e registrazione del componente
- Come rendere pubbliche le proprietà ereditate
- Cambiamento dei valori iniziali
- Ridimensionamento delle celle
- Riempimento delle celle
- Scorrimento di mesi e anni
- Scorrimento dei giorni

Nelle applicazioni VCL, il componente che viene creato è simile al componente *TCalendar* della pagina Sample della Component palette. Nelle applicazioni CLX, salvare il componente in una pagina differente oppure creare una nuova pagina della tavolozza. Vedere "Specifica della pagina della tavolozza" a pagina 52-3 oppure "Component palette, adding pages" nella guida in linea.

Creazione e registrazione del componente

La creazione di qualsiasi componente inizia sempre nello stesso modo: si deriva la classe di un componente, si salvano il file .CPP e .H del componente, lo si registra, lo si compila e lo si installa nella Component palette. Questo processo viene descritto nella sezione "Creazione di un nuovo componente" a pagina 45-8.

Per questo esempio, è necessario seguire la procedura generale per la creazione di un componente, con le seguenti specifiche:

1 Derivare un nuovo componente di nome *TSampleCalendar*, discendente da *TCustomGrid*.

2 Assegnare come nome al file header del componente CLSAMP.H e al file .CPP associato CALSAMP.CPP.

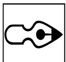
3 Registrare *TSampleCalendar* nella pagina Samples (o in un'altra pagina della CLX) della Component palette. Nel caso di applicazioni CLX, utilizzare una pagina differente della Component palette.

Il file header apparirà più o meno in questo modo:

```
#ifndef CalSampH
#define CalSampH
//---------------------------------------------------------------------------
#include <vcl\sysutils.hpp>
#include <vcl\controls.hpp>
#include <vcl\classes.hpp>
#include <vcl\forms.hpp>
#include <vcl\grids.hpp>
//---------------------------------------------------------------------------
class PACKAGE TSampleCalendar : public TCustomGrid
{
private:
protected:
public:
__published:
};
//---------------------------------------------------------------------------
#endif
```

Il file CALSAMP.CPP sarà simile al seguente:

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "CalSamp.h"
//---------------------------------------------------------------------------
#pragma package(smart_init);
//---------------------------------------------------------------------------
static inline TSampleCalendar *ValidCtrCheck()
{
  return new TSampleCalendar(NULL);
}
//---------------------------------------------------------------------------
namespace Calsamp
{
  void __fastcall PACKAGE Register()
  {
    TComponentClass classes[1] = {__classid(TSampleCalendar)};
    RegisterComponents("Samples", classes, 0); //In CLX, use a different page than "Samples"
  }
}
```



Se per creare il componente si è utilizzato il Component wizard componente, il file header conterrà anche la dichiarazione di un nuovo costruttore e il file CALSAMP.CPP conterrà le istruzioni iniziali di un costruttore. Se non c'è nessun costruttore, è possibile aggiungerlo in seguito.

Nelle applicazioni CLX, i nomi e le ubicazioni di alcuni file header sono differenti. Ad esempio, <vcl\controls.hpp> diventa <clx\qcontrols.hpp> in CLX.

Come rendere pubbliche le proprietà ereditate

Il componente griglia astratto, *TCustomGrid*, è provvisto di un grande numero di proprietà dichiarate **protected**. È possibile scegliere quali di queste proprietà rendere disponibili per l’utente del controllo calendario.

Per rendere disponibili agli utenti le proprietà protette ereditate dal componente, è necessario dichiararle nella sezione **_published** della dichiarazione del componente.

Per il controllo calendario bisogna rendere pubbliche le seguenti proprietà ed eventi:

```
class PACKAGE TSampleCalendar : public TCustomGrid
{
⋮
__published:
    __property Align ;                  // publish properties
    __property BorderStyle ;
    __property Color ;
    __property Font ;
    __property GridLineWidth ;
    __property ParentColor ;
    __property ParentFont ;
    __property OnClick ;                 // publish events
    __property OnDblClick ;
    __property OnDragDrop ;
    __property OnDragOver ;
    __property OnEndDrag ;
    __property OnKeyDown ;
    __property OnKeyPress ;
    __property OnKeyUp ;
};
```

Ci sono numerose altre proprietà che possono anche essere rese pubbliche, ma che non vengono utilizzate nel calendario, come *Options* che abilita l’utente a scegliere il tipo di linea da utilizzare nella griglia del disegno.

Se si installa il componente calendario modificato nella Component palette e si utilizza in un’applicazione, nel calendario si troveranno molte proprietà ed eventi disponibili, tutti completamente funzionali. A questo punto è possibile aggiungere nuove possibilità al proprio progetto.

Cambiamento dei valori iniziali

Un calendario è essenzialmente una griglia con un numero definito di righe e di colonne e inoltre non tutte le righe contengono sempre dei dati. Per questo motivo non sono state rese pubbliche le proprietà *ColCount* e *RowCount*, dal momento che è molto improbabile che gli utenti del calendario vogliano visualizzare più di sette

giorni alla settimana. Si deve anche fissare il valore iniziale di queste proprietà, in modo da avere sempre sette giorni alla settimana.

Per modificare i valori iniziali delle proprietà del componente, ridefinire il costruttore per impostare i valori desiderati.

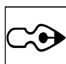
Occorre ricordarsi di aggiungere il costruttore nella parte **public** della dichiarazione della classe del componente e di scrivere il nuovo costruttore nel file header:

```
class PACKAGE TSampleCalendar : public TCustomGrid
{
protected:
    virtual void __fastcall DrawCell(int ACol, int ARow, const Windows::TRect &Rect,
      TGridDrawState AState);
    ⋮
public:
    __fastcall TSampleCalendar(TComponent *Owner);   // the added constructor
    ⋮
};
```

Nel file CALSAMP.CPP scrivere il codice del costruttore:

```
__fastcall TSampleCalendar::TSampleCalendar(TComponent *Owner) : TCustomGrid(Owner)
{
  ColCount = 7;
  RowCount = 7;
  FixedCols = 0;
  FixedRows = 1;
  ScrollBars = ssNone;
  Options = (Options >> goRangeSelect) << goDrawFocusSelected;
}

void __fastcall TSampleCalendar::DrawCell(int ACol, int ARow, const Windows::TRect
  &ARect, TGridDrawState AState)
{
}
```



Si può notare che alla dichiarazione della classe è stato anche aggiunto un metodo *DrawCell*, e che nel file .CPP è stato iniziato il metodo *DrawCell*. Ciò non è assolutamente necessario ora, ma se si cercasse di collaudare *TSampleCalendar* prima di ridefinire *DrawCell*, si incapperebbe in un errore legato alla funzione pure virtual. Ciò è dovuto al fatto che *TCustomGrid* è una classe astratta.La ridefinizione di *DrawCell* viene discussa più avanti, nella sezione "Riempimento delle celle".

Il calendario ora ha sette colonne e sette righe, di cui la riga superiore è fissa o senza funzionalità di scorrimento.

Ridimensionamento delle celle

Quando un utente o un'applicazione cambia le dimensioni di una finestra o di un controllo, Windows invia un messaggio di nome *WM_SIZE* alla finestra o al controllo in questione, in modo da poter sistemare qualsiasi impostazione necessaria per il successivo disegno nella nuova dimensione. Il componente VCL può rispondere a questo messaggio modificando le dimensioni delle celle in modo che tutte le celle

possano essere contenute entro i limiti del controllo. Per rispondere al messaggio *WM_SIZE*, è necessario aggiungere al componente un metodo per la gestione dei messaggi.

La creazione di tale metodo è descritta in maniera più dettagliata nel paragrafo "Creazione di nuovi gestori di messaggio" a pagina 51-6.

In questo caso, poiché il controllo calendario deve rispondere a *WM_SIZE*, occorre aggiungere al controllo un metodo protetto di nome *WMSize*, quindi scrivere il metodo in modo tale che possa calcolare la grandezza necessaria delle celle permettendo alle stesse di essere visibili nella nuova dimensione:

```
class PACKAGE TSampleCalendar : public TCustomGrid
{
⋮
protected:
    void __fastcall WMSize(TWMSize &Message);

BEGIN_MESSAGE_MAP
  MESSAGE_HANDLER(WM_SIZE, TWMSize, WMSize)
END_MESSAGE_MAP(TCustomGrid)
};
```

Ecco il codice per il metodo nel file CALSAMP.CPP:

```
void __fastcall TSampleCalendar::WMSize(TWMSize &Message)
{
  int GridLines;                                  // temporary local variable
  GridLines = 6 * GridLineWidth;                  // calculated combined size of all lines
  DefaultColWidth = (Message.Width - GridLines) / 7;    // set new default cell width
  DefaultRowHeight = (Message.Height - GridLines) / 7;  // and cell height
}
```

A questo punto, quando si ridimensiona il calendario, esso visualizza tutte le celle utilizzando la massima ampiezza possibile per riempire completamente il controllo.

In CLX, le modifiche apportate alle dimensioni di una finestra o di un controllo vengono immediatamente notificate tramite una chiamata al metodo protetto *BoundsChanged*. Il componente CLX può rispondere a questa notifica alterando le dimensioni delle celle in modo che tutte le celle siano contenute entro i margini del controllo.

In questo caso, il controllo calendario deve ridefinire *BoundsChanged* in modo tale che sia in grado di calcolare la grandezza delle celle necessaria affinché esse restino visibili pur assumendo nuove dimensioni:

```
class PACKAGE TSampleCalendar : public TCustomGrid
{
⋮
protected:
    void __fastcall BoundsChanged(void);
};
```

Ecco il codice per il metodo nel file CALSAMP.CPP :

```
void __fastcall TSampleCalendar::BoundsChanged(void)
{
  int GridLines;                                  // temporary local variable
```

```
  GridLines = 6 * GridLineWidth;                  // calculated combined size of all lines
  DefaultColWidth = (Width - GridLines) / 7;    // set new default cell width
  DefaultRowHeight = (Height - GridLines) / 7;  // and cell height
  TCustomGrid::BoundsChanged(); // now call the inherited method
}
```

Riempimento delle celle

Un controllo griglia inserisce il proprio contenuto cella per cella. Nel caso del calendario, ciò significa calcolare quale data, se esiste, appartiene a ogni cella. Il tracciamento predefinito delle celle della griglia viene effettuato in un metodo virtuale di nome *DrawCell*.

Per riempire le celle della griglia, ridefinire il metodo *DrawCell*.

La parte più facile da riempire è quella delle celle dell'intestazione nella riga fissa. Poiché la libreria runtime contiene già un array con i nomi abbreviati dei giorni, per il calendario usare quello adatto per ciascuna colonna:

Questo è il codice per il metodo *DrawCell*:

```
void __fastcall TSampleCalendar::DrawCell(int ACol, int ARow, const Windows::TRect &ARect,
  TGridDrawState AState)
{
  String TheText;
  int TempDay;
  if (ARow == 0) TheText = ShortDayNames[ACol + 1];
  else
  {
    TheText = "";
    TempDay = DayNum(ACol, ARow);                   // DayNum is defined later
    if (TempDay != -1) TheText = IntToStr(TempDay);
  }
  Canvas->TextRect(ARect, ARect.Left + (ARect.Right - ARect.Left
   - Canvas->TextWidth(TheText)) / 2,
  ARect.Top + (ARect.Bottom - ARect.Top - Canvas->TextHeight(TheText)) / 2, TheText);
}
```

Registrazione della data

Affinché il controllo del calendario risulti utile, gli utenti e le applicazioni devono avere un meccanismo di impostazione del giorno, del mese e dell'anno. C++Builder registra le date e le ore in variabili di tipo *TDateTime*. *TDateTime* è una rappresentazione numerica codificata della data e dell'ora, utile per la gestione tramite programma ma poco pratica per un utilizzo da parte di un individuo.

È quindi possibile registrare la data in una forma codificata, consentendo l'accesso a tale valore in esecuzione, e fornire anche le proprietà *Day*, *Month*, e *Year* in modo che l'utente del componente calendario possa impostarle in fase di progettazione.

La registrazione della data è un processo composto dalle seguenti fasi:

- Registrazione della data interna
- Accesso a giorno, mese e anno
- Generazione dei numeri dei giorni
- Selezione del giorno corrente

Registrazione della data interna

Per registrare la data per il calendario, è necessario un membro dati privato in cui memorizzare la data e alcune proprietà, attive solo in esecuzione, che consentono l'accesso a tale data.

1 Dichiarare un membro dati privato per memorizzare la data:

```
class PACKAGE TSampleCalendar : public TCustomGrid
{
private:
    TDateTime FDate;
    ⋮
};
```

2 Inizializzare nel costruttore la data del membro dati:

```
__fastcall TSampleCalendar::TSampleCalendar(TComponent *Owner) : TCustomGrid(Owner)
{
  ⋮
  FDate = FDate.CurrentDate();
}
```

3 Dichiarare una proprietà di runtime per permettere l'accesso alla data codificata:

```
class PACKAGE TSampleCalendar : public TCustomGrid
{
public:
    __property TDateTime CalendarDate = {read=FDate, write=SetCalendarDate, nodefault};
    ⋮
};
```

È necessario un metodo per l'impostazione della data, poiché tale operazione deve anche aggiornare l'immagine a video del controllo. Dichiarare *SetCalendarDate* in *TSampleCalendar*:

```
class PACKAGE TSampleCalendar : public TCustomGrid
{
private:
    void __fastcall SetCalendarDate(TDateTime Value);
    ⋮
};
```

Questo è il metodo *SetCalendar*:

```
void __fastcall TSampleCalendar::SetCalendarDate(TDateTime Value)
{
  FDate = Value;                      // Set the new date value
  Refresh();                          // Update the onscreen image
}
```

Accesso a giorno, mese e anno

Una data codificata numericamente è adatta per le applicazioni, ma è più comodo lavorare con giorni, mesi e anni. È possibile preparare accessi alternativi a questi elementi codificati creando delle proprietà.

Poiché ogni elemento della data (giorno, mese, anno) è un intero e poiché l'impostazione di ognuno richiede la ricodifica della data, è possibile evitare la duplicazione del codice ogni volta, condividendo il metodo per tutte e tre le proprietà. È possibile scrivere due metodi, uno per leggere un elemento e uno per scriverlo, e utilizzare questi metodi per acquisire e impostare le tre proprietà.

Per poter accedere a giorno, mese e anno durante lo sviluppo, bisogna passare per le seguenti fasi:

1 Dichiarare le tre proprietà, assegnando a ognuna un numero di indice univoco:

```
class PACKAGE TSampleCalendar : public TCustomGrid
{
  ⋮
public:
    __property int Day = {read=GetDateElement, write=SetDateElement, index=3,
      nodefault};
    __property int Month = {read=GetDateElement, write=SetDateElement, index=2,
      nodefault};
    __property int Year = {read=GetDateElement, write=SetDateElement, index=1,
      nodefault};
};
```

2 Dichiarare e scrivere il metodo di implementazione, impostando i diversi elementi per ogni valore dell'indice:

```
class PACKAGE TSampleCalendar : public TCustomGrid
{
private:
    int __fastcall GetDateElement(int Index);           // note the Index parameter
    void __fastcall SetDateElement(int Index, int Value);
    ⋮
};
```

Ecco i metodi *GetDateElement* e *SetDateElement*:

```
int __fastcall TSampleCalendar::GetDateElement(int Index)
{
  unsigned short AYear, AMonth, ADay;
  int result;
  FDate.DecodeDate(&AYear, &AMonth, &ADay);            // break encoded date into elements
  switch (Index)
  {
    case 1: result = AYear;  break;
    case 2: result = AMonth; break;
    case 3: result = ADay;   break;
    default: result = -1;
  }
  return result;
}
```

```
void __fastcall TSampleCalendar::SetDateElement(int Index, int Value)
```

```
{
  unsigned short AYear, AMonth, ADay;
  if (Value > 0)                                    // all elements must be positive
  {
    FDate.DecodeDate(&AYear, &AMonth, &ADay);     // get current date elements
    switch (Index)
    {
      case 1: AYear = Value;     break;
      case 2: AMonth = Value;    break;
      case 3: ADay = Value;      break;
      default: return;
    }
  }
  FDate = TDateTime(AYear, AMonth, ADay);         // encode the modified date
  Refresh();                                      // update the visible calendar
}
```

Ora è possibile impostare giorno, mese e anno del calendario in fase di progetto utilizzando l'Object Inspector oppure in fase di esecuzione tramite programma. Naturalmente non è stato ancora aggiunto il codice per disegnare le date nelle celle, ma ora è disponibile la data desiderata.

Generazione dei numeri dei giorni

L'inserimento dei numeri nel calendario implica diverse considerazioni. Il numero di giorni in un mese dipende da quale mese si considera e bisogna ricordarsi degli anni bisestili. Inoltre, il primo giorno del mese cade in giorni della settimana differenti, a seconda del mese e dell'anno. Se l'anno è bisestile, utilizzare la funzione *IsLeapYear*. Per ottenere il numero dei giorni di un mese, usare l'array *MonthDays* nel file header SysUtils.

Quando sono note le informazioni sugli anni bisestili e sul numero di giorni in ogni mese, è possibile calcolare dove inserire la singola data sulla griglia. Il calcolo è basato sul giorno della settimana con cui comincia il mese.

Poiché per ogni cella da riempire sarà necessario conoscere il numero di scostamento del mese, il metodo migliore è quello di calcolarne il valore ogni volta che vengono cambiati il mese o l'anno, e quindi fare riferimento ogni volta a tale valore. È possibile registrare il valore in un membro dati della classe, quindi aggiornare tale membro dati ogni volta che cambia la data.

Per inserire i giorni nelle celle appropriate:

1 Aggiungere un membro dati per l'offset del mese alla classe e un metodo che aggiorni il valore del membro dati:

```
class PACKAGE TSampleCalendar : public TCustomGrid
{
private:
    int FMonthOffset;                          // storage for the offset
    ⋮
protected:
    virtual void __fastcall UpdateCalendar(void);
    ⋮
};
```

```
void __fastcall TSampleCalendar::UpdateCalendar(void)
{
  unsigned short AYear, AMonth, ADay;
  TDateTime FirstDate;                          // date of first day of the month
  if ((int)FDate != 0)                          // only calculate offset if date is valid
  {
    FDate.DecodeDate(&AYear, &AMonth, &ADay);  // get elements of date
    FirstDate = TDateTime(AYear, AMonth, 1);   // date of the first
    FMonthOffset = 2 - FirstDate.DayOfWeek();  // generate the offset into the grid
  }
  Refresh();                                    // always repaint the control
}
```

2 Aggiungere istruzioni al costruttore e ai metodi *SetCalendarDate* e *SetDateElement* che chiamino il nuovo metodo di aggiornamento ogni volta che si cambia la data:

```
__fastcall TSampleCalendar::TSampleCalendar(TComponent *Owner)
  : TCustomGrid(Owner)
{
  ⋮
  UpdateCalendar();
}

void __fastcall TSampleCalendar::SetCalendarDate(TDateTime Value)
{
  FDate = Value;                                  // this was already here
  UpdateCalendar();                               // this previously called Refresh
}

void __fastcall TSampleCalendar::SetDateElement(int Index, int Value)
{
  ⋮
  FDate = TDateTime(AYear, AMonth, ADay);        // this was already here
  UpdateCalendar();                              // this previously called Refresh
}
```

3 Aggiungere al calendario un metodo che restituisca il numero del giorno quando vengono passate le coordinate della riga e della colonna di una cella:

```
int __fastcall TSampleCalendar::DayNum(int ACol, int ARow)
{
  int result = FMonthOffset + ACol + (ARow - 1) * 7;       // calculate day for this cell
  if ((result < 1)||(result > MonthDays[IsLeapYear(Year)][Month]))
    result = -1;   // return -1 if invalid
  return result;
}
```

Ricordarsi di aggiungere la dichiarazione di *DayNum* alla dichiarazione di tipo del componente.

4 Ora che è possibile calcolare dove deve essere inserita la data, è necessario aggiornare *DrawCell* per inserire le date:

```
void __fastcall TSampleCalendar::DrawCell(int ACol, int ARow, const TRect &ARect,
  TGridDrawState AState)
{
```

```
  String TheText;
  int TempDay;
  if (ARow == 0)                                   // this is the header row
    TheText = ShortDayNames[ACol + 1];             // just use the day name
  else
  {
    TheText = "";                                  // blank cell is the default
    TempDay = DayNum(ACol, ARow);                  // get number for this cell
    if (TempDay != -1) TheText = IntToStr(TempDay); // use the number if valid
  }
  Canvas->TextRect(ARect, ARect.Left + (ARect.Right - ARect.Left -
    Canvas->TextWidth(TheText)) / 2,
  ARect.Top + (ARect.Bottom - ARect.Top - Canvas->TextHeight(TheText)) / 2, TheText);
}
```

Se ora si reinstalla il componente calendario e lo si inserisce in una scheda, si possono vedere le informazioni corrette per il mese corrente.

Selezione del giorno corrente

Ora che sono stati inseriti i numeri nelle celle del calendario, sarebbe opportuno poter muovere la selezione evidenziando la cella contenente il giorno attuale. Per impostazione predefinita, la selezione inizia nella cella in alto a sinistra, pertanto bisogna impostare le proprietà di *Row* e *Column* sia quando si costruisce inizialmente il calendario sia quando si cambia la data.

Per impostare la selezione sul giorno corrente, cambiate il metodo *UpdateCalendar* per impostare *Row* e *Column* prima di chiamare *Refresh*:

```
void __fastcall TSampleCalendar::UpdateCalendar(void)
{
  unsigned short AYear, AMonth, ADay;
  TDateTime FirstDate;
  if ((int) FDate != 0)
  {
  ⋮                                   // existing statements to set FMonthOffset
    Row = (ADay - FMonthOffset) / 7 + 1;
    Col = (ADay - FMonthOffset) % 7;
  }
  Refresh();                          // this is already here
}
```

Si noti che è stata riutilizzata la variabile *Aday* impostata in precedenza per decodificare la data.

Scorrimento di mesi e anni

Le proprietà sono utili per la gestione dei componenti, specialmente in fase di sviluppo. Qualche volta però ci sono tipi di operazioni che sono comuni o naturali e che magari coinvolgono più di una proprietà, tanto da suscitare la necessità di sviluppare un metodo per gestirli. Un esempio è la possibilità di passare al

successivo mese nel calendario. La gestione dello scorrimento dei mesi e l'incremento degli anni è semplice e molto utile per gli utilizzatori del componente.

L'unico svantaggio nell'incapsulare le operazioni più comuni nei metodi è che i metodi sono disponibili solo in fase di esecuzione. Comunque, tali operazioni sono piuttosto pesanti solo se effettuate ripetutamente, il che in fase di progettazione è molto raro.

Per il calendario aggiungere i seguenti quattro metodi per il mese e l'anno precedente e successivo. Ognuno di questi metodi utilizza la funzione *IncMonth* con una leggera differenza per aumentare o diminuire *CalendarDate*, con incrementi di un mese o di un anno.

```
void __fastcall TSampleCalendar::NextMonth()
{
  CalendarDate = IncMonth(CalendarDate, 1);
}
void __fastcall TSampleCalendar::PrevMonth()
{
  CalendarDate = IncMonth(CalendarDate, -1);
}
void __fastcall TSampleCalendar::NextYear()
{
  CalendarDate = IncMonth(CalendarDate, 12);
}

void __fastcall TSampleCalendar::PrevYear()
{
  CalendarDate = IncMonth(CalendarDate, -12);
}
```

Assicurarsi di aggiungere le dichiarazioni del nuovo metodo alla dichiarazione della classe.

Ora, quando si crea un'applicazione che utilizza il componente calendario, è possibile implementare lo scorrimento dei mesi o degli anni.

Scorrimento dei giorni

Una volta noto il mese, esistono due modi ovvi per spostarsi tra i giorni. Il primo è quello di utilizzare le frecce, l'altro è quello di usare il mouse. I componenti standard della griglia li utilizzano entrambi come se fossero dei clic. Cioè, un movimento con la freccia è trattato come se fosse un clic su una cella.

Il processo di scorrimento dei giorni è composta dalle seguenti fasi:

- Spostamento della selezione
- Preparazione di un evento OnChange
- Esclusione delle celle vuote

Spostamento della selezione

Il comportamento ereditato di una griglia gestisce lo spostamento della selezione in risposta sia ai tasti freccia sia ai clic, ma per cambiare il giorno selezionato è necessario modificare tale comportamento.

Per cambiare i movimenti all'interno del calendario, ridefinire il metodo *Click* della griglia.

Quando si ridefinisce un metodo come *Click*, che è vincolato alle interazioni dell'utente, bisogna quasi sempre includere una chiamata al metodo ereditato, così da non perdere il funzionamento predefinito.

La procedura seguente ridefinisce il metodo *Click* per una griglia di calendario. Assicurarsi di aggiungere la dichiarazione di *Click* in *TSampleCalendar*.

```
void __fastcall TSampleCalendar::Click()
{
  int TempDay = DayNum(Col, Row);           // get the day number for the clicked cell
  if (TempDay != -1) Day = TempDay;         // change day if valid
}
```

Preparazione di un evento OnChange

Ora che l'utente del calendario è in grado di cambiare la data, è giusto permettere alle applicazioni di rispondere a questi cambiamenti.

Add an *OnChange* event to *TSampleCalendar*.

1 Dichiarare l'evento, un data member per memorizzare l'evento e un metodo virtual per poterlo chiamare:

```
class PACKAGE TSampleCalendar : public TCustomGrid
{
private:
    TNotifyEvent FOnChange;
    ⋮
protected:
    virtual void __fastcall Change();
__published:
    __property TNotifyEvent OnChange = {read=FOnChange, write=FOnChange};
    ⋮
}
```

2 Scrivere il metodo *Change*:

```
void __fastcall TSampleCalendar::Change()
{
  if(FOnChange != NULL) FOnChange(this);
}
```

3 Aggiungere le istruzioni per la chiamata a *Change* alla fine dei metodi *SetCalendarDate* e *SetDateElement*:

```
void __fastcall TSampleCalendar::SetCalendarDate(TDateTime Value)
{
```

```
    FDate = Value;
    UpdateCalendar();
    Change();              // this is the only new statement
  }

  void __fastcall TSampleCalendar::SetDateElement(int Index, int Value)
  {
    ⋮   // many statements setting element values
    FDate = TDateTime(AYear, AMonth, ADay);
    UpdateCalendar();
    Change();              // this is new
  }
```

Le applicazioni che utilizzano il componente calendario sono ora in grado di rispondere ai cambiamenti della data del componente collegando codice di gestione all'evento *OnChange*.

Esclusione delle celle vuote

Una volta scritto il calendario, l'utente può selezionare una cella vuota, ma la data non cambia. È quindi utile disattivare la selezione delle celle vuote.

Per controllare se una cella è selezionabile, ridefinire il metodo *SelectCell* nella griglia.

SelectCell è una funzione che accetta come parametri una riga e una colonna e restituisce un valore Boolean che indica se la cella specificata è selezionabile.

È possibile ridefinire *SelectCell* restituendo **false** se la cella non contiene una data valida:

```
bool __fastcall TSampleCalendar::SelectCell(int ACol, int ARow)
{
  if (DayNum(ACol,ARow) == -1) return false;          // -1 indicates invalid date
  else return TCustomGrid::SelectCell(ACol, ARow);    // otherwise, use inherited value
}
```

Se ora l'utente fa clic su una cella vuota o tenta di muoversi su una cella vuota con le frecce, il calendario lascia selezionata la cella corrente.

Capitolo

56

Creazione di un controllo di accesso ai dati

Quando si lavora con connessioni database, spesso è conveniente avere controlli associati ai dati. Ovvero, l'applicazione può stabilire un collegamento tra il controllo e una parte del database. C++Builder include etichette, caselle di testo, caselle di riepilogo, caselle combinate, controlli di consultazione e griglie associati ai dati. È anche possibile creare propri controlli associati ai dati. Per ulteriori informazioni sui controlli *data-aware*, consultare il Capitolo 19, "Uso dei controlli dati".

Esistono diversi livelli di associazione ai dati. Il più semplice è un controllo per accedere a dati in sola lettura, o *data browsing*, utilizzato per controllare lo stato attuale di un database. Più complesso è il controllo relativo a dati che si possono anche modificare, o *data editing*, dove l'utente può scrivere i valori da riportare nel database gestendo il controllo. Inoltre, si noti che il livello di interfaccia con un database può variare: nel caso più semplice si ha un collegamento con un singolo campo, in casi più complessi si sviluppa un controllo che ha accesso a più record.

Per prima cosa il capitolo illustra il caso più semplice, cioè la creazione di un controllo a sola lettura che si collega a un singolo campo in un dataset. Il controllo specifico da utilizzare è *TSampleCalendar*, creato nel Capitolo 55, "Personalizzazione di una griglia". È anche possibile utilizzare il controllo calendario standard, *TCalendar*, presente nella pagina Samples della Component palette.

Quindi il capitolo continua con una spiegazione di come trasformare il nuovo controllo per la consultazione dei dati in un controllo per l'*editing* dei dati.

Creazione di un controllo per l'esame dei dati

La creazione di un controllo calendario associato ai dati, sia che sia un controllo a sola lettura o uno nel quale l'utente può modificare i dati associati del dataset, coinvolge le seguenti operazioni:

Creazione e registrazione del componente

La creazione di qualsiasi componente inizia sempre nello stesso modo: si deriva la classe di un componente, si salvano il file .CPP e .H del componente, lo si registra, lo si compila e lo si installa nella Component palette. Questo processo viene descritto nella sezione “Creazione di un nuovo componente” a pagina 45-8.

Per questo esempio seguire la procedura generale per la creazione di un componente, con le seguenti specifiche:

- Derivare una nuova classe del componente di nome *TDBCalendar*, discendente da *TSampleCalendar*. Capitolo 55, “Personalizzazione di una griglia”, mostra come creare il componente *TSampleCalendar*.
- Assegnare come nome DBCAL.H al file haeder e DBCAL.CPP al file .CPP.
- Registrare *TDBCalendar* nella pagina Samples (o in un'altra pagina della CLX) della Component palette.

Nelle applicazioni CLX i nomi e le ubicazioni di alcuni file header sono differenti. Ad esempio, <vcl\controls.hpp> è <clx\qcontrols.hpp> in CLX.

Il file header risultante apparirà più o meno come il seguente:

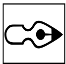
```
#ifndef DBCalH
#define DBCalH
//---------------------------------------------------------------------------
#include <vcl\sysutils.hpp>
#include <vcl\controls.hpp>
#include <vcl\classes.hpp>
#include <vcl\forms.hpp>
#include <vcl\grids.hpp>              // include the Grids header
#include “calsamp.h”                  // include the header that declares TSampleCalendar
//---------------------------------------------------------------------------
class PACKAGE TDBCalendar : public TSampleCalendar
{
private:
protected:
public:
__published:
};
//---------------------------------------------------------------------------
#endif
```

Il file .CPP risulterà simile a questo:

```
#pragma link “Calsamp”                       // link in TSampleCalendar

#include <vcl\vcl.h>
#pragma hdrstop
#include "DBCal.h"
```

```
//---------------------------------------------------------------------------
#pragma package(smart_init);
//---------------------------------------------------------------------------
static inline TDBCalendar *ValidCtrCheck()
{
   return new TDBCalendar(NULL);
}
//---------------------------------------------------------------------------
namespace Dbcal
{
  void __fastcall PACKAGE Register()
  {
    TComponentClass classes[1] = {__classid(TDBCalendar)};
    RegisterComponents("Samples", classes, 0); //Use different page in CLX applications
  }
}
```



Se si è utilizzato il wizard Component per iniziare la creazione del componente *TDBCalendar*, il file header conterrà il costruttore già dichiarato e il file .CPP avrà la definizione del costruttore.

Ora è possibile trasformare il nuovo componente in un visualizzatore di dati.

Costruzione del controllo a sola lettura

Dal momento che questo calendario sarà a sola lettura, ha senso definire il controllo stesso come a sola lettura, in modo che gli utenti non possano effettuarvi cambiamenti e ritrovarli nei dati del database.

Definire un calendario a sola lettura implica:

- Aggiunta della proprietà ReadOnly.
- Permettere gli aggiornamenti necessari.

Si noti che, iniziando con il componente *TCalendar* della pagina Samples di C++Builder, anziché con *TSampleCalendar*, è già presente la proprietà *ReadOnly* , quindi questi passi possono essere saltati.

Aggiunta della proprietà ReadOnly

Aggiungendo una proprietà *ReadOnly* , è possibile introdurre un metodo che garantisca che il controllo sia accessibile in sola lettura in fase di sviluppo. Quando questa proprietà è impostata a **true**, è possibile rendere non selezionabili tutte le celle incluse nel controllo.

1 Aggiungere al file DBCAL.H la dichiarazione della proprietà e un membro dati **private** per registrarne il valore:

```
class PACKAGE TDBCalendar : public TSampleCalendar
{
private:
    bool FReadOnly;                                  // field for internal storage
protected:
public:
```

```
    virtual __fastcall TDBCalendar(TComponent* Owner);
__published:
    __property ReadOnly = {read=FReadOnly, write=FReadOnly, default=true};
};
```

2 Scrivere il costruttore nel file DBCAL.CPP:

```
virtual __fastcall TDBCalendar::TDBCalendar(TComponent* Owner) :
  TSampleCalendar(Owner)
{
  FReadOnly = true;                                   // sets the default value
}
```

3 Ridefinire il metodo *SelectCell* per disabilitare la selezione se il controllo è a sola lettura. L'utilizzo di *SelectCell* viene illustrato in "Esclusione delle celle vuote" a pagina 55-14.

```
bool __fastcall TDBCalendar::SelectCell(long ACol, long ARow)
{
  if (FReadOnly) return false;                        // can't select if read only
  return TSampleCalendar::SelectCell(ACol, ARow);     // otherwise, use inherited method
}
```

Bisogna ricordarsi di aggiungere la dichiarazione *SelectCell* alla dichiarazione della classe di *TDBCalendar*.

Se si aggiunge il calendario a una scheda, il componente ignora i clic e le pressioni dei tasti. Commette un errore anche nell'aggiornare la posizione selezionata quando si cambia la *data*.

Permettere gli aggiornamenti necessari

Il nuovo calendario utilizza il metodo *SelectCell* per tutti i tipi di cambiamenti, incluse le impostazioni delle proprietà *Row* e *Col*. Il metodo *UpdateCalendar* imposta *Row* e *Col* ogni volta che viene cambiata la *data*, ma dal momento che *SelectCell* non permette i cambiamenti, la selezione rimane dov'era anche se la *data* cambia.

Per aggirare questa proibizione assoluta sui cambiamenti, è possibile aggiungere un flag interno di tipo Boolean al calendario e permettere i cambiamenti quando questo flag è impostato a **true**:

```
class PACKAGE TDBCalendar : public TSampleCalendar
{
private:
    ⋮
    bool FUpdating;                                  // private flag for internal use
protected:
    virtual bool __fastcall SelectCell(long ACol, long ARow);
public:
    ⋮
    virtual void __fastcall UpdateCalendar();
    ⋮
};
```

```
bool __fastcall TDBCalendar::SelectCell(long ACol, long ARow)
{
    if (!FUpdating && FReadOnly) return false;          // can't select if read only
```

```
    return TSampleCalendar::SelectCell(ACol, ARow);     // otherwise, use inherited method
}

void __fastcall TDBCalendar::UpdateCalendar()
{
  FUpdating=true;                                      // set flag to allow updates
  try
  {
     TSampleCalendar::UpdateCalendar();               // update as usual
  }
  catch(...)
  {
    FUpdating = false;
    throw;
  }
  FUpdating = false;                                   // always clear the flag
}
```

Il calendario non permette ancora i cambiamenti da parte dell'utente, ma ora riflette correttamente i cambiamenti della *data* effettuati cambiando le proprietà. Ora che si ha un corretto controllo a sola lettura, è possibile aggiungere la connessione alla base dati.

Aggiunta di un datalink

Il collegamento tra un controllo e un database è gestito da una classe chiamata datalink. La classe per il collegamento dati che collega un controllo con un singolo membro dati di un database è *TFieldDataLink.* Ci sono inoltre collegamenti di dati per intere tabelle.

Un controllo associato ai dati possiede la sua classe datalink. Questo significa che il controllo ha la responsabilità della costruzione e della distruzione del datalink. Per maggiori informazioni sulla gestione delle classi possedute, consultare il Capitolo 54, "Creazione di un controllo grafico".

Per creare un datalink come una classe posseduta, effettuare queste tre operazioni:

1. Dichiarazione del membro dati.
2. Dichiarzione delle proprietà di accesso.
3. Inizializzazione del datalink.

Dichiarazione del membro dati

Come spiegato nella "Dichiarazione dei membri dati della classe" a pagina 54-6 un componente ha bisogno di un membro dati per ciascuna delle classi che possiede. In questo caso, per il collegamento dei dati, il calendario ha bisogno di un membro dati di tipo *TFieldDataLink*.

Dichiarare un campo per il collegamento dei dati nel calendario:

```
class PACKAGE TDBCalendar : public TSampleCalendar
{
private:
```

```
    TFieldDataLink *FDataLink;
    ⋮
};
```

Prima di compilare l'applicazione, è necessario includere i file DB.HPP e DBTABLES.HPP nel file DBCAL.H:

```
#include <DB.hpp>
#include <DBTables.hpp>
```

Dichiarzione delle proprietà di accesso

Ogni controllo *data-aware* ha una proprietà *DataSource* che specifica quale classe datasource nell'applicazione fornisce i dati al controllo. Inoltre, un controllo che accede ad un singolo campo ha bisogno della proprietà *DataField* per specificare quel campo nella sorgente dati.

A differenza di quanto visto nell'esempio relativo alle classi possedute riportato nel Capitolo 54, "Creazione di un controllo grafico", le proprietà di accesso trattate nell'esempio non consentono di accedere alle classi possedute, ma alle proprietà corrispondenti nella classe posseduta. In pratica, bisogna costruire proprietà che abilitano il controllo e il relativo collegamenti a condividere la stessa sorgente dati e lo stesso campo.

Dichiarare le proprietà *DataSource* e *DataField* e i rispettivi metodi di implementazione, quindi scrivere i metodi come metodi "pass-through" per arrivare alle proprietà corrispondenti della classe datalink:

Un esempio di dichiarazione delle proprietà di accesso

```
class PACKAGE TDBCalendar : public TSampleCalendar
{
private:
   ⋮
    AnsiString __fastcall GetDataField();             // methods are private
    TDataSource *__fastcall GetDataSource();          // returns name of data field
    void __fastcall SetDataField(AnsiString Value); // returns reference to data
                                                       // source
    void __fastcall SetDataSource(TDataSource *Value);    // assigns name of data field
    ⋮
__published:                                 // make properties available at design time
    __property AnsiString DataField = {read=GetDataField, write=SetDataField, nodefault};
    __property TDataSource * DataSource = {read=GetDataSource, write=SetDataSource,
      nodefault};
   ⋮
};

AnsiString __fastcall TDBCalendar::GetDataField()
{
  return FDataLink->FieldName;
}

TDataSource *__fastcall TDBCalendar::GetDataSource()
{
  return FDataLink->DataSource;
}
```

```
void __fastcall TDBCalendar::SetDataField(AnsiString Value)
{
  FDataLink->FieldName = Value;
}
void __fastcall TDBCalendar::SetDataSource(TDataSource *Value)
{
  if(Value != NULL)
    Value->FreeNotification(this);
  FDataLink->DataSource = Value;
}
```

Ora che sono stati stabiliti i collegamenti tra il calendario e il relativo datalink, c'è il passo più importante. È necessario costruire una classe datalink quando il controllo calendario viene generato e distruggere il datalink prima della distruzione del calendario.

Inizializzazione del datalink

Un controllo associato ai dati deve poter accedere al proprio datalink finché esiste e quindi deve costruire l'oggetto datalink all'interno del proprio costruttore e distruggerlo prima di essere a sua volta distrutto.

Ridefinire il costruttore e il distruttore del calendario:

```
class PACKAGE TDBCalendar : public TSampleCalendar
{
public:
    virtual __fastcall TDBCalendar(TComponent *Owner);
    __fastcall ~TDBCalendar();
};
__fastcall TDBCalendar::TDBCalendar(TComponent* Owner) : TSampleCalendar(Owner)
{
    FReadOnly = true;
    FDataLink = new TFieldDataLink();
    FDataLink->Control = this;
}
__fastcall TDBCalendar::~TDBCalendar()
{
    FDataLink->Control = NULL;
    FDataLink->OnUpdateData = NULL;
    delete FDataLink;
}
```

A questo punto è stato completato il collegamento ai dati, ma non è stato ancora specificato al controllo quali dati deve leggere dal campo collegato. La prossima sezione spiega come fare.

Risposta alle modifiche dei dati

Una volta che un controllo è stato dotato di un proprio datalink e di proprietà che consentono di specificare la sorgente dati e il campo dati, deve poter rispondere ai cambiamenti dei dati in tale campo, sia dovuti a uno spostamento in un record

diverso sia per reagire a un cambiamento effettuato sul valore del campo all'interno della base dati.

Tutte le classi datalink dispongono di eventi chiamati *OnDataChange*. Quando il *data*-sorce indica un cambiamento nei propri dati, l'oggetto datalink chiama il gestore di evento collegato al proprio evento *OnDataChange*.

Per aggiornare un controllo in risposta a un cambiamento di dati, si aggiunge un gestore all'evento *OnDataChange* del datalink.

In questo caso, si può aggiungere un metodo al calendario, quindi indicarlo come il gestore per l'evento *OnDataChange* del datalink.

Dichiarare e implementare il metodo *DataChange*, quindi assegnarlo all'evento *OnDataChange* del datalink nel costruttore. Nel distruttore, prima di distruggere l'oggetto, sconnettere il gestore *OnDataChange*.

```
class PACKAGE TDBCalendar : public TSampleCalendar
{
private:
    void __fastcall DataChange(TObject *Sender);
    ⋮
};

void __fastcall TDBCalendar::DataChange( TObject* Sender)
{
  if (FDataLink->Field == NULL)                          // if no field is assigned ...
  CalendarDate = 0;                                      // ...set to invalid date
  else CalendarDate = FDataLink->Field->AsDateTime;  // otherwise, set to new data
}

__fastcall TDBCalendar::TDBCalendar(TComponent* Owner) :  TSampleCalendar(AOwner)
{
  FReadOnly = true;
  FDataLink = new TFieldDataLink();            // construct the datalink object
  FDataLink->Control = this;
  FDataLink->OnDataChange = DataChange;        // attach the handler
}

__fastcall TDBCalendar::~TDBCalendar()
{
  FDataLink->Control = NULL;
  FDataLink->OnUpdateData = NULL;
  FDataLink->OnDataChange = NULL;                 // detach the handler before...
  delete FDataLink;                               // ...destroying the datalink object
}
```

Ora il controllo per l'esame dei dati è pronto.

Creazione di un controllo per l'editing dei dati

Quando si crea un controllo per la modifica di dati si crea e registra il componente e si aggiunge il datalink esattamente nello stesso modo utilizzato per un controllo per la consultazione dei dati. In modo analogo, occorre rispondere anche alle modifiche apportate ai dati nel campo associato ma occorre gestire qualche problema ulteriore.

Per esempio, è probabile che il controllo debba risponda sia agli eventi della tastiera sia a quelli del mouse. Il controllo deve rispondere quando l'utente ne modifica i contenuti. Quando l'utente esce dal controllo, le modifiche apportate nel controllo dovranno essere riportate nel dataset.

Il controllo per la modifica di dati descritto di seguito è lo stesso controllo calendario descritto nella prima parte del capitolo. Il controllo viene modificato in modo da poter modificare e visualizzare i dati nel campo collegato.

La modifica di un controllo esistente per trasformarlo in un controllo per la modifica di dati comporta le seguenti operazioni:

- Modifica del valore predefinito di FReadOnly.
- Gestione dei messaggi mouse-down e key-down.
- Modifica della classe datalink del campo.
- Modifica del metodo Change.
- Aggiornamento del dataset.

Modifica del valore predefinito di FReadOnly

Poiché questo è un controllo per la modifica di dati, per impostazione predefinita la proprietà *ReadOnly* dovrebbe essere impostata a **false**. Per rendere **false** la proprietà *ReadOnly*, modificare nel costruttore il valore *FReadOnly*:

```
__fastcall TDBCalendar::TDBCalendar (TComponent* Owner) : TSampleCalendar(Owner)
{
  FReadOnly = false;            // set the default value
  ⋮
}
```

Gestione dei messaggi mouse-down e key-down

Quando l'utente del controllo comincia ad interagire con esso, il controllo riceve o un messaggio mouse-down (*WM_LBUTTONDOWN*, *WM_MBUTTONDOWN*, o *WM_RBUTTONDOWN*) oppure un messaggio key-down (*WM_KEYDOWN*) da Windows. Per fare in modo che un controllo risponda a questi messaggi, si devono scrivere appositi gestori di messaggio.

- Risposta ai messaggi mouse-down
- Risposta a messaggi key-down

Nel caso si usi CLX, la notifica proviene dal sistema operativo sotto forma di eventi di sistema. Per informazioni sulla scrittura di componenti che rispondono a aventi di sistema e di widget, consultare "Risposta alle notifiche di sistema utilizzando la CLX" a pagina 51-11.

Risposta ai messaggi mouse-down

Un metodo *MouseDown* è un metodo protetto dell'evento *OnMouseDown* di un controllo. Il controllo stesso chiama *MouseDown* in risposta a un messaggio mouse-down di Windows. Quando si ridefinisce il metodo *MouseDown* ereditato, si può

includere codice che fornisce altre risposte oltre a quella di chiamata dell'evento *OnMouseDown*.

Per ridefinire *MouseDown*, aggiungere un metodo *MouseDown* alla classe *TDBCalendar*:

```
class PACKAGE TDBCalendar : public TSampleCalendar
{
⋮
protected:
    virtual void __fastcall MouseDown(TMouseButton Button,  TShiftState Shift, int X,
      int Y);
    ⋮
};
```

Scrivere il metodo *MouseDown* nel file .CPP:

```
void __fastcall TDBCalendar::MouseDown(TMouseButton Button,  TShiftState Shift, int  X,
  int Y)
{
  TMouseEvent MyMouseDown;                                 // declare event type
  if (!FReadOnly && FDataLink->Edit())                     // if the field can be edited
    TSampleCalendar::MouseDown(Button, Shift, X, Y);   // call the inherited MouseDown
  else
  {
    MyMouseDown = OnMouseDown;                            // assign OnMouseDown event
    if (MyMouseDown != NULL) MyMouseDown(this, Button,  // execute code in the...
       Shift, X, Y);                                      // ...OnMouseDown event handler
  }
}
```

Quando *MouseDown* risponde a un messaggio del mouse, il metodo *MouseDown* ereditato viene chiamato solamente se la proprietà *ReadOnly* del controllo è **false** e l'oggetto datalink è in modalità modifica, il che significa che il campo può essere modificato. Se il campo non può essere modificato, viene eseguito il codice inserito dal programmatore nel gestore evento *OnMouseDown*, se ne esiste uno.

Risposta a messaggi key-down

Il metodo *KeyDown* è un metodo protetto per l'evento *OnKeyDown* di un controllo. Il controllo stesso chiama *KeyDown* in risposta a un messaggio key-down di Windows. Quando si ridefinisce il metodo *MouseDown* ereditato, si può includere del codice che fornisca altre risposte oltre a quella di chiamata dell'evento *OnMouseDown*.

Per ridefinire *KeyDown*, seguire questa procedura:

1 Aggiungere un metodo *KeyDown* alla classe *TDBCalendar*:

```
class PACKAGE TDBCalendar : public TSampleCalendar
{
   ⋮
protected:
    virtual void __fastcall KeyDown(unsigned short &Key,  TShiftState Shift);
   ⋮
};
```

2 Scrivere il metodo *KeyDown* nel file .CPP:

```
void __fastcall TDBCalendar::KeyDown(unsigned short &Key,  TShiftState Shift)
{
  TKeyEvent MyKeyDown;                              // declare event type
  Set<unsigned short,0,8> keySet;
  keySet = keySet << VK_UP << VK_DOWN << VK_LEFT      // assign virtual keys to set
    << VK_RIGHT << VK_END << VK_HOME << VK_PRIOR << VK_NEXT;
  if (!FReadOnly &&                               // if control is not read only...
      (keySet.Contains(Key)) &&                    // ...and key is in the set...
      FDataLink->Edit() )                          // ...and field is in edit mode
  {
    TCustomGrid::KeyDown(Key, Shift);          // call the inherited KeyDown method
  }
  else
  {
    MyKeyDown = OnKeyDown;                          // assign OnKeyDown event
    if (MyKeyDown != NULL) MyKeyDown(this,Key,Shift); // execute code in...
  }                                                    // ...OnKeyDown event handler
}
```

Quando *KeyDown* risponde a un messaggio key-down, il metodo *KeyDown* ereditato viene chiamato solamente se la proprietà *ReadOnly* del controllo è **false**, il tasto premuto è uno dei tasti di controllo del cursore e l'oggetto del datalink è in modalità modifica, il che significa che il campo può essere modificato. Se il campo non può essere modificato, viene eseguito il codice inserito dal programmatore nel gestore di evento *OnKeyDown*, se ne esiste uno.

Modifica della classe datalink del campo

Ci sono due tipi di modifiche dei dati:

- Una modifica di un valore di campo che deve essere riportata nel controllo associato ai dati.
- Una modifica del controllo associato ai dati che deve essere riportata nel valore di campo.

Il componente *TDBCalendar* ha già un metodo *DataChange* che gestisce una modifica del valore del campo nel dataset assegnando tale valore alla proprietà *CalendarDate*. Il metodo *DataChange* è il gestore dell'evento *OnDataChange*. Quindi, il componente calendario può gestire il primo tipo di modifica dei dati.

Analogamente, anche la classe datalink del campo ha un evento *OnUpdateData* che si verifica quando l'utente del controllo modifica i contenuti del controllo *data-aware*. Il controllo calendario ha un metodo *UpdateData* che diviene il gestore dell'evento *OnUpdateData*. *UpdateData* assegna il valore modificato del controllo *data-aware* al campo *data-link*.

1 Per riflettere una modifica effettuata al valore nel calendario nel valore del campo, aggiungere un metodo *UpdateData* alla sezione **private** del componente calendario:

```
class PACKAGE TDBCalendar : public TSampleCalendar
{
private:
```

```
    void __fastcall UpdateData(TObject *Sender);
};
```

2 Scrivere il metodo *UpdateData* nel file .CPP:

```
void __fastcall TDBCalendar::UpdateData( TObject* Sender)
{
    FDataLink->Field->AsDateTime = CalendarDate;     // set field link to calendar date
}
```

3 All'interno del costruttore per *TDBCalendar* assegnare il metodo *UpdateData* all'evento *OnUpdateData*:

```
__fastcall TDBCalendar::TDBCalendar(TComponent* Owner)
  : TSampleCalendar(Owner)
{
  FDataLink = new TFieldDataLink();       // this was already here
  FDataLink->OnDataChange = DataChange;   // this was here too
  FDataLink->OnUpdateData = UpdateData;   // assign UpdateData to the OnUpdateData event
}
```

Modifica del metodo Change

Il metodo *Change* di *TDBCalendar* viene chiamato ogni qualvolta si imposta un nuovo valore della *data*. *Change* chiama il gestore di evento *OnChange*, se ne esiste uno. L'utente del componente può scrivere codice nel gestore di evento *OnChange* per rispondere alle modifiche della *data*.

Quando varia la *data* del calendario, ai dataset associati si dovrebbe notificare la variazione. Questo si può fare ridefinendo il metodo *Change* e aggiungendo un'istruzione ulteriore. Questi sono i passi da seguire:

1 Aggiungere un nuovo metodo *Change* al componente *TDBCalendar*:

```
class PACKAGE TDBCalendar : public TSampleCalendar
{
protected:
    virtual void __fastcall Change();
    ⋮
};
```

2 Scrivere il metodo *Change*, chiamando il metodo *Modified* che informa il dataset che i dati sono stati modificati, quindi chiamare il metodo *Change* ereditato:

```
void __fastcall TDBCalendar::Change()
{
  if (FDataLink != NULL)
    FDataLink->Modified();              // call the Modified method
  TSampleCalendar::Change();            // call the inherited Change method
}
```

Aggiornamento del dataset

Finora, una modifica all'interno del controllo *data-aware* ha modificato valori nella classe datalink del campo. Il passo finale della creazione di un controllo modifica dati è quello di aggiornare il dataset col nuovo valore. L'aggiornamento dovrebbe essere effettuato quando chi ha modificato il valore nel controllo associato ai dati esce dal controllo facendo clic esternamente al controllo o premendo il tasto Tab. Questo processo funziona in modo diverso nella VCL e nella CLX.

Nella VCL sono definiti vari ID per il controllo dei messaggi relativi alle operazioni sui controlli. Per esempio, il messaggio *CM_EXIT* viene trasmesso al controllo quando l'utente ne esce. Si possono scrivere gestori di messaggi che rispondono al messaggio. In questo caso, quando l'utente esce dal controllo, il metodo *CMExit*, il gestore del messaggio *CM_EXIT*, risponde aggiornando il record nel dataset coi valori modificati nella classe campo datalink. Per maggiori informazioni sui gestori di messaggio, consultare il Capitolo 51, "Gestione dei messaggi e delle notifiche di sistema".

Per aggiornare il dataset all'interno di un gestore messaggi:

1 Aggiungere il gestore messaggio al componente *TDBCalendar*:

```
class PACKAGE TDBCalendar : public TSampleCalendar
{
private:
    void __fastcall CMExit(TWMNoParams Message);

BEGIN_MESSAGE_MAP
  MESSAGE_HANDLER(CM_EXIT, TWMNoParams, CMExit)
END_MESSAGE_MAP
};
```

2 Scrivere il codice nel file .CPP in modo che somigli al seguente:

```
void __fastcall TDBCalendar::CMExit(TWMNoParams &Message)
{
  try
  {
    FDataLink.UpdateRecord();          // tell data link to update database
  }
  catch(...)
  {
    SetFocus();                        // if it failed, don't let focus leave
    throw;
  }
}
```

Nella CLX, *TWidgetControl* ha un metodo protetto *DoExit* che viene chiamato quando il fuoco viene spostato via dal controllo. Questo metodo chiama il gestore di evento dell'evento *OnExit*. È possibile ridefinire questo metodo per aggiornare il record nel dataset prima di generare il gestore di evento *OnExit*.

Per aggiornare il dataset quando l'utente si sposta dal controllo, fare quanto segue:

1 Aggiungere la direttiva override al metodo *DoExit* del componente *TDBCalendar*:

```
class PACKAGE TDBCalendar : public TSampleCalendar
{
private:
    DYNAMIC void __fastcall DoExit(void);
   ⋮
};
```

2 Scrivere il codice nel file .CPP in modo che somigli al seguente:

```
void __fastcall TDBCalendar::DoExit(void)
{
  try
  {
    FDataLink.UpdateRecord();          // tell data link to update database
  }
  catch(...)
  {
    SetFocus();                        // if it failed, don't let focus leave
    throw;
  }
  TCustomGrid::DoExit(); // let the inherited method generate an OnExit event
}
```

Capitolo

57

Trasformazione di una finestra di dialogo in un componente

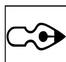
A volte potrebbe risultare utile trasformare una finestra di dialogo usata di frequente in un componente da aggiungere alla Component palette. I componenti finestra di dialogo funzioneranno esattamente come i componenti standard delle finestre di dialogo comuni. Lo scopo è quello di creare un semplice componente che possa essere aggiunto a un progetto e per il quale sia possibile impostarne le proprietà in fase di progettazione.

La trasformazione di una finestra di dialogo in un componente richiede le seguenti operazioni:

1 Definizione dell'interfaccia del componente.
2 Creazione e registrazione del componente
3 Creazione dell'interfaccia del componente
4 Verifica del componente

Il componente "wrapper" di C++Builder associato alla finestra di dialogo, crea ed esegue la finestra di dialogo durante l'esecuzione, passando i dati specificati dall'utente. Il componente finestra di dialogo risulta pertanto riutilizzabile e Personalizzazione.

Questo capitolo descrive come creare un componente contenitore (wrapper) che contiene la finestra About Box fornita nell'Object Repository di C++Builder.



Copiare i file ABOUT.H, ABOUT.CPP e ABOUT.DFM nella directory di lavoro. Aggiungere ABOUT.CPP al progetto in modo da creare un file ABOUT.OBJ quando si costruisce il componente contenitore.

Non ci sono considerazioni speciali da fare per la progettazione di una finestra di dialogo che deve essere inglobata in un componente. Quasi tutte le schede possono operare come una finestra di dialogo in questo contesto.

Definizione dell'interfaccia del componente.

Prima di creare il componente per la finestra di dialogo, occorre decidere come svilupparlo in modo da poterlo utilizzare. È possibile creare un'interfaccia tra la finestra di dialogo e le applicazioni che la utilizzano.

Per esempio, si osservino le proprietà dei componenti di una comune finestra di dialogo. È possibile notare che consentono agli sviluppatori di impostare lo stato iniziale della finestra di dialogo, come il titolo e l'impostazione iniziale dei controlli, e, una volta chiusa la finestra di dialogo, di rileggere tutte le informazioni necessarie. Non ci sono interazioni dirette con i singoli controlli nella finestra di dialogo, ma solo con le proprietà nel componente contenitore.

L'interfaccia deve inoltre contenere informazioni sufficienti affinché la scheda della finestra di dialogo possa apparire nella modalità specificata dallo sviluppatore e restituisca ogni informazione richiesta dall'applicazione. Si può pensare alle proprietà incapsulate nel componente contenitore come a dati persistenti per una finestra di dialogo transiente.

La finestra About non deve restituire alcuna informazione, così le proprietà del contenitore devono contenere solo le informazioni necessarie per visualizzare correttamente la finestra. Poiché ci sono quattro campi separati nella finestra About che possono essere modificati dalle applicazioni, si forniranno, per questi, quattro proprietà di tipo stringa.

Creazione e registrazione del componente

La creazione di qualsiasi componente inizia sempre nello stesso modo: si deriva la classe di un componente, si salvano il file .CPP e .H del componente, lo si registra, lo si compila e lo si installa nella component palette. Questo processo viene descritto nella sezione "Creazione di un nuovo componente" a pagina 45-8.

Per questo esempio è necessario seguire la procedura generale per la creazione di un componente, con le seguenti specifiche:

- Derivare un nuovo tipo di componente chiamato *TAboutBoxDlg*, discendente da *TComponent*.
- Chiamare i file unit del componente ABOUTDLG.H e ABBOUTDLG.CPP.
- Registrare *TAboutBoxDlg* nella pagina Sample della Component palette.

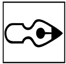
Il file .HPP risultante dovrebbe essere simile al seguente:

```
#ifndef AboutDlgH
#define AboutDlgH
//---------------------------------------------------------------------------
#include <vcl\sysutils.hpp>
#include <vcl\controls.hpp>
#include <vcl\classes.hpp>
#include <vcl\forms.hpp>
//---------------------------------------------------------------------------
```

```
class PACKAGE TAboutBoxDlg : public TComponent
{
private:
protected:
public:
__published:
};
//---------------------------------------------------------------------------
#endif
```

Il file .CPP della unit sarà simile al seguente:

```
#include <vcl\vcl.h>
#pragma hdrstop
#include "AboutDlg.h"
//---------------------------------------------------------------------------
#pragma package(smart_init);
//---------------------------------------------------------------------------
static inline TAboutBoxDlg *ValidCtrCheck()
{
  return new TAboutBoxDlg(NULL);
}
//---------------------------------------------------------------------------
namespace AboutDlg {
{
  void __fastcall PACKAGE Register()
  {
    TComponentClass classes[1] = {__classid(TAboutBoxDlg)};
    RegisterComponents("Samples", classes, 0);
  }
}
```



Se è stato utilizzato il wizard Component per inizializzare questo componente, a *TAboutDlg* verrà aggiunto anche un costruttore.

Nelle applicazioni CLX i nomi e le ubicazioni di alcuni file header sono differenti. Ad esempio, <vcl\controls.hpp> è <clx\qcontrols.hpp> in CLX.

In questa fase il nuovo componente ha solamente le capacità intrinseche ed ereditate da *TComponent*. È il più semplice componente non visuale. Nella sezione successiva si creerà l’interfaccia tra il componente e la finestra di dialogo.

Creazione dell’interfaccia del componente

Queste sono le operazioni da svolgere per creare l’interfaccia del componente:

1. Inclusione dei file unit della scheda.
2. Aggiunta delle proprietà di interfaccia.
3. Aggiunta del metodo Execute.

Inclusione dei file unit della scheda

Per inizializzare il proprio componente wrapper e visualizzare la finestra di dialogo inclusa, è necessario aggiungere i file della scheda al progetto.

Includere ABOUT.H e il collegamento ad ABOUT.OBJ nel file header del componente:

```
#include "About.h"
#pragma link "About.obj"
```

Il file header della scheda dichiara sempre un'istanza della classe della scheda. Nel caso della finestra About, la classe della scheda è *TAboutBox* e il file ABOUT.H include quanto segue:

```
extern TAboutBox *AboutBox;
```

Aggiunta delle proprietà di interfaccia

Prima di procedere, bisogna decidere quali sono le proprietà necessarie al contenitore affinché gli sviluppatori possano utilizzare la finestra di dialogo come componente nelle applicazioni. Per ottenere ciò, si possono aggiungere le dichiarazioni per queste proprietà nella dichiarazione del componente.

Le proprietà nei componenti contenitore sono qualcosa di più semplice delle proprietà che si potrebbero creare scrivendo un semplice componente. Ricordarsi che, in questo caso, si stanno creando alcuni dati persistenti che il contenitore può passare alla finestra di dialogo e ricevere da essa. Inserendo questi dati sotto forma di proprietà, si consente agli sviluppatori di impostare i dati in fase di progetto, in modo che il contenitore li possa passare alla finestra di dialogo in fase di esecuzione.

La dichiarazione di una proprietà di interfaccia richiede due aggiunte alla dichiarazione della classe del componente:

- Un membro dati privato, cioè la variabile che il contenitore usa per registrare il valore della proprietà
- La dichiarazione published stessa della proprietà, che specifica il nome della proprietà e indica quale membro dati utilizzare per la registrazione.

Le proprietà di interfaccia di questo tipo non necessitano di metodi di accesso. Utilizzano un accesso diretto ai propri dati registrati. Per convenzione, il membro dati che contiene il valore di proprietà ha lo stesso nome della proprietà, ma preceduto dalla lettera *F*. Il membro dati e la proprietà devono essere dello stesso tipo.

Per esempio, per dichiarare una proprietà di interfaccia di tipo integer di nome *Year*, dichiararla come segue:

```
class PACKAGE TWrapper : public TComponent
{
private:
    int FYear;                              // data member to hold the Year-property data
protected:
```

```
public:
__published:
    __property int Year = {read=FYear, write=FYear};        // property matched with storage
};
```

Per la finestra About servono quattro proprietà di tipo string, rispettivamente per il nome del prodotto, l'informazione sulla versione, l'informazione sul copyright e un eventuale commento. Il file ABOUTDLG.H apparirà ora in questo modo:

```
class PACKAGE TAboutBoxDlg : public TComponent
{
private:
    int FYear;
    String FProductName, FVersion, FCopyright, FComments;
protected:
public:
__published:
    __property int Year = {read=FYear, write=FYear};
    __property String ProductName = {read=FProductName, write=FProductName};
    __property String Version = {read=FVersion, write=FVersion};
    __property String Copyright = {read=FCopyright, write=FCopyright};
    __property String Comments = {read=FComments, write=FComments};
};
```

Quando si installa il componente nella component Palette e lo si colloca su una scheda, sarà possibile impostarne le proprietà, e questi valori saranno automaticamente inseriti nella scheda. Il contenitore potrà quindi utilizzare questi valori quando esegue la finestra di dialogo inglobata.

Aggiunta del metodo Execute

La parte finale del componente interfaccia è un metodo per aprire la finestra di dialogo e restituire un risultato quando viene chiusa. Come con i componenti finestra di dialogo comune, è possibile utilizzare una funzione booleana di nome *Execute* che restituisce **true** se l'utente fa clic su OK, oppure **false** se l'utente annulla la finestra di dialogo.

La dichiarazione del metodo *Execute* è sempre simile alla seguente:

```
class PACKAGE TMyWrapper : public TComponent
{
  ⋮
public:
    bool __fastcall Execute();
  ⋮
};
```

L'implementazione minima di *Execute* deve costruire la finestra di dialogo, mostrarla come una finestra di dialogo modale e restituire **true** o **false**, a seconda del valore restituito da *ShowModal*.

Questo è il metodo *Execute* per una finestra di dialogo di tipo *TMyDialogBox*:

```
bool __fastcall TMyWrapper::Execute()
{
```

```
  DialogBox = new TMyDialogBox(Application);    // construct the form
    bool Result;
  try
  {
    Result = (DialogBox->ShowModal()  IDOK);  // execute; set result based on how closed
  }
  catch(...)
  {
    Result = false;                             // if it fails, set Result to false
  }
  DialogBox->Free();                            // dispose of form
}
```

In pratica, ci sarà più codice all’interno di exception handler. In particolare, prima della chiamata *ShowModal*, il contenitore imposta alcune proprietà della finestra di dialogo basandosi sulle proprietà di interfaccia del componente contenitore. Al ritorno da *ShowModal*, è probabile che il contenitore imposti alcune proprietà di interfaccia in base al risultato dell’esecuzione della finestra di dialogo.

Nel caso della finestra About, sarà necessario utilizzare le quattro proprietà di interfaccia del componente contenitore per impostare il contenuto delle etichette nella finestra della scheda About. Poiché la finestra About non restituisce nessuna informazione all’applicazione, non c’è la necessità di fare qualcosa dopo la chiamata a *ShowModal*. Scrivere il metodo *Execute* del wrapper della finestra About in modo che il file ABOUTDLG.CPP sia simile al seguente:

```
bool __fastcall TAboutBoxDlg::Execute()
{
  AboutBox = new TAboutBox(Application);              // construct the About box
  bool Result;
  try
  {
    if (ProductName == “”)                            // if product name’s left blank ...
      ProductName = Application->Title;               // ... use application title instead
    AboutBox->ProductName->Caption = ProductName;     // copy product name
    AboutBox->Version->Caption = Version;             // copy version information
    AboutBox->Copyright->Caption = Copyright;         // copy copyright information
    AboutBox->Comments->Caption = Comments;           // copy comments
    AboutBox->Caption = “About “ + ProductName;       // set About-box caption
    Result = (AboutBox->ShowModal() == IDOK);         // execute and set result
  }
  catch(...)
  {
    Result = false;                      // if it fails, set Result to false
    ⋮
  }
  AboutBox->Free();                      // dispose of About box
  return Result == IDOK;                 // compare Result to IDOK and return Boolean value
}
```

Aggiungere alla parte public della classe *TAboutDlg* nel file header ABOUTDLG.H la dichiarazione per il metodo *Execute*:

```
class PACKAGE TAboutDlg : public TComponent
{
```

```
public:
    virtual bool __fastcall Execute();
};
```

Verifica del componente

Una volta installato il componente della finestra di dialogo, lo si può usare come si fa con le comuni finestre di dialogo, mettendone uno su una scheda ed eseguendo la scheda. Un modo rapido per verificare la finestra di dialogo About è quello di aggiungere un pulsante a una scheda che mandi in esecuzione la finestra di dialogo quando l'utente fa clic sul pulsante.

Ad esempio, se si è creato una finestra di dialogo About, la si è trasformata in un componente e la si è aggiunta alla Component palette, è possibile collaudarla nel seguente modo:

1. Creare un nuovo progetto.
2. Collocare un componente About sulla scheda principale.
3. Inserire un pulsante nella scheda.
4. Fare doppio clic sul pulsante per scrivere il gestore di evento clic.
5. Inserire la seguente linea di codice:

   ```
   AboutBoxDlg1->Execute();
   ```

6. Eseguire l'applicazione.

Quando appare la scheda principale, fare clic sul pulsante. Apparirà la finestra About con l'icona predefinita del progetto e il nome Project1. Scegliere OK per chiudere la finestra di dialogo.

Si può collaudare ulteriormente il componente impostando le varie proprietà del componente About ed eseguendo nuovamente l'applicazione.

Capitolo

58

Estensione dell'IDE

È possibile estendere e personalizzare l' IDE con voci di menu, pulsanti della barra strumenti, wizard dinamici per la creazione di schede e altro, utilizzando le API di Open Tools (spesso abbreviato in API Tools). Le API Tools sono un insieme di più di 100 interfacce (classi astratte) che interagiscono con e controllano l'IDE , compresi il menu principale, le barre strumenti, l'elenco delle azioni principali e delle immagini, i buffer dell'editor interno dei sorgenti, le macro e i collegamenti di tastiera, le schede e i relativi componenti nell'editor delle schede, il debugger e il processo in fase di debug, il completamento del codice, la vista dei messaggi e la lista To-Do.

Utilizzare le API Tools significa semplicemente scrivere classi che implementano alcune interfacce e chiamare i servizi forniti da altre interfacce. Il codice delle API Tools deve essere compilato e caricato nell'IDE in fase di progettazione come un package di progettazione oppure in una DLL. Quindi, scrivere un'estensione delle API Tools è più o meno come scrivere un editor di proprietà o di un componente. Prima di affrontare questo capitolo, assicurarsi di avere familiarità con le nozioni di base per operare con i package (Capitolo 15, "Uso di package e di componenti") e per registrare componenti (Capitolo 52, "Come rendere disponibili i componenti in fase di progetto"). Sarebbe anche bene leggere il Capitolo 13, "Supporto del linguaggio C++ per la VCL e la CLX", soprattutto la sezione "Eredità e interfacce" a pagina 13-2 per informazioni sulle interfacce in stile Delphi.

Questo capitolo tratta i seguenti argomenti:

- Panoramica sugli strumenti API
- Scrittura di una classe wizard
- Ottenimento dei servizi delle API Tools
- Operazioni con file ed editor
- Creazione di schede e di progetti
- Notifica a un wizard di eventi dell'IDE
- Installazione della DLL di un wizard

Panoramica sugli strumenti API

Tutte le dichiarazioni delle API Tools risiedono in un singolo file header, ToolsAPI.hpp; il namespace è Toolsapi. Di solito, per utilizzare le API Tools, si usa il package designide: è cioè necessario costruire gli add-in delle API Tools come un package di progettazione o come una DLL che utilizza package di esecuzione. Per informazioni sui problemi realtivi ai package e alle librerie, vedere "Installazione del package del wizard" a pagina 58-7 and "Installazione della DLL di un wizard" a pagina 58-23.

L'interfaccia principale per la scrittura di un'estensione delle API Tools è *IOTAWizard*, ed è per questo che la maggior parte degli add-in dell'IDE sono detti wizard. La maggior parte dei wizard di C++Builder e di Delphi sono intercambiabili. È possibile scrivere e compilare un wizard in Delphi e utilizzarlo in seguito in C++Builder, e viceversa. L'interoperabilità funziona al meglio con lo stesso numero di versione, ma è anche possibile scrivere wizard da utilizzare in future versioni di entrambi i prodotti. Per ulteriori informazioni sulla compatibilità con future versioni, vedere "Utilizzo di una DLL senza package di esecuzione" a pagina 58-24.

Per utilizzare le API Tools, vengono scritte le classi dei wizard che implementano una o più interfacce definite nella unit ToolsAPI. Si ricordi dal Capitolo 13 che C++Builder rappresenta un'interfaccia Object Pascal come una classe astratta. Per implementare l'interfaccia, è necessario ridefinire le funzioni membro della classe astratta e i suoi antenati e implementare la funzione QueryInterface per riconoscere l'interfaccia GUID.

Un wizard utilizza i servizi forniti dalle API Tools. Ogni servizio è costituito da un'interfaccia che presenta un insieme di funzioni connesse. L'implementazione dell'interfaccia è nascosta all'interno dell'IDE. Le API Tools pubblicano solo l'interfaccia, che può essere utilizzata per scrivere i wizard senza occuparsi dell'implementazione delle interfacce. I vari servizi permettono l'accesso all'editor dei sorgenti, al designer delle schede, al debugger e così via. La sezione "Ottenimento dei servizi delle API Tools" a pagina 58-8 esamina questo argomento in modo approfondito.

Il servizio e le altre interfacce rientrano in due categorie di base. È possibile distinguerle grazie al prefisso utilizzato per il nome del tipo:

- NTA (native tools API) garantisce l'accesso diretto agli oggetti effettivi IDE, come l'oggetto *TMainMenu* dell'IDE. Quando vengono utilizzate queste interfacce, il wizard deve usare i package di Borland, il che significa anche che il wizard è legato a una versione specifica dell'IDE. Il wizard può risiedere in un package di progettazione o in una DLL che utilizza i package di esecuzione.
- OTA (open tools API) non richiedere package e accede all'IDE solo attraverso le interfacce. In teoria, si potrebbe scrivere un wizard in qualsiasi linguaggio che supporta le interfacce in stile COM, a condizione che sia possibile operare anche con la convenzione di chiamata __fastcall di Borland e con i tipi Object Pascal come AnsiString. Le interfacce OTA non garantiscono il completo accesso all'IDE, ma quasi tutte le funzionalità delle API Tools sono disponibili attraverso le

interfacce OTA. Se un wizard utilizza solo le interfacce OTA, è possibile scrivere una DLL che non dipenda da una versione specifica dell'IDE.

Le API Tools hanno due tipi di interfacce: un tipo che deve essere implementato dal programmatore e un tipo che viene implementato dall'IDE. La maggior parte delle interfacce appartiene a quest'ultima categoria: le interfacce definiscono la capacità dell'IDE ma nascondono l' implementazione corrente. I tipi di interfacce che devono essere implementate ricadono in tre categorie: wizard, notifier e creator:

- Come è già stato menzionato in questa sezione, una classe wizard implementa l'interfaccia *IOTAWizard* e le eventuali interfacce derivate.
- Un notifier è un altro tipo di interfaccia delle API Tools. L'IDE utilizza il notifier per richiamare il wizard quando accade qualcosa di interessante. Il programmatore scrive una classe che implementa l'interfaccia notifier e registra il notifier con le API Tools e l'IDE esegue una call back all'oggetto notifier quando l'utente apre un file, modifica il codice sorgente, modifica una scheda, avvia una sessione di debug, e così via. I notifier vengono trattati in "Notifica a un wizard di eventi dell'IDE" a pagina 58-19.
- Un creator è un altro tipo di interfaccia che è necessario implementare. Le API Tools utilizzano un creator per creare nuove unit, progetti o altri file o per aprire i file esistenti. La sezione "Creazione di schede e di progetti" a pagina 58-14 tratta questi argomenti in modo più approfondito.

Altre importanti interfacce sono i moduli e gli editor. Un'interfaccia di modulo rappresenta una unit aperta, che ha uno o più file. Un'interfaccia di editor rappresenta un file aperto. Diversi tipi di interfacce editor consentono l'accesso a diversi aspetti dell'IDE: l'editor dei sorgente ai file sorgente, il Form designer ai file scheda e le risorse di progetto ai file di risorse. La sezione "Operazioni con file ed editor" a pagina 58-13 tratta questi argomenti in modo approfondito.

Le seguenti sezioni descrivono le fasi per la scrittura di un wizard. Fare riferimento ai file della guida in linea per i dettagli completi di ogni interfaccia.

Scrittura di una classe wizard

Ci sono quattro tipi di wizard, che si differenziano per le interfacce implementate dalla classe wizard. La Tabella 58.1 descrive i quattro tipi di wizard.

Tabella 58.1 I quattro tipi di wizard

Interfaccia	Descrizione
IOTAFormWizard	Di solito crea una nuova unit, scheda o altro file
IOTAMenuWizard	Viene automaticamente aggiunto al menu della guida
IOTAProjectWizard	Di solito crea una nuova applicazione o un altro progetto
IOTAWizard	Wizard eterogeneo che non rientrano in altre categorie

I quattro tipi di wizard si differenziano solo per il modo in cui il wizard viene richiamato dall'utente:

- Un menu wizard viene aggiunto al menu Help dell'IDE. Quando l'utente sceglie la voce di menu, l'IDE chiama la funzione Execute() del wizard. I wizard normali offrono una maggiore flessibilità, perciò i menu wizard vengono di solito utilizzati solo per prototipi e per eseguire il debug.
- I wizard di scheda e i wizard di progetto vengono chiamati wizard repository perché risiedono nell'Object Repository. L'utente chiama questi wizard dalla finestra di dialogo New Items. L'utente può anche vedere i wizard nell'Object Repository (scegliendo la voce di menu Tools| Repository). L'utente può selezionare la casella di controllo New Form di un wizard di scheda, la quale comunica all'IDE di chiamare il wizard di scheda quando l'utente sceglie la voce di menu File|New Form. L'utente può inoltre selezionare la casella di controllo Main Form. La casella di controllo dice all'IDE di impiegare il wizard delle schede come scheda predefinita per una nuova applicazione. L'utente può controllare la casella di controllo New Project per un wizard di progetto. Quando l'utente sceglie File|New Application, l'IDE chiama il wizard di progetto selezionato.
- Il quarto tipo di wizard viene utilizzato in situazioni che non rientrano nelle altre categorie. Un normale wizard non fa niente automaticamente o da solo. Al contario, è necessario definire il modo in cui il wizard viene richiamato.

Le API Tools non impongono nessuna restrizione al wizard, come richiedere un wizard di progetto per creare un progetto. È possibile scrivere semplicemente un wizard di progetto per creare una scheda e un wizard di schede per creare un progetto (se è qualcosa che si desidera realmente fare).

Implementazione delle interfacce wizard

Ogni classe wizard deve implementare almeno *IOTAWizard*, che richiede di implementare anche i suoi antenati: *IOTANotifier* e *IInterface*. I wizard delle schede e dei progetti devono implementare tutte le loro interfacce antenato, cioè, *IOTARepositoryWizard*, *IOTAWizard*, *IOTANotifier* e *IInterface*.

L'implementazione di *IInterface* deve seguire le normali regole delle interfacce Object Pascal, che sono le stesse di quelle delle interfacce COM. Vale a dire, *QueryInterface* esegue conversioni di tipo e *AddRef* e *Release* gestiscono il conteggio dei riferimenti. Si potrebbe volere utilizzare una classe base comune per semplificare la scrittura del wizard e delle classi notifier.

Ad esempio, Delphi ha la classe *TNotifierObject*, che implementa *IOTANotifier* con corpi di funzione vuoti. È possibile scrivere una classe simile in C++, come verrà mostrato in seguito.

```
class PACKAGE NotifierObject : public IOTANotifier {
public:
  __fastcall NotifierObject() : ref_count(0) {}
  virtual __fastcall ~NotifierObject();
  void __fastcall AfterSave();
  void __fastcall BeforeSave();
  void __fastcall Destroyed();
  void __fastcall Modified();
protected:
```

```
    // IInterface
    virtual HRESULT __stdcall QueryInterface(const GUID&, void**);
    virtual ULONG __stdcall AddRef();
    virtual ULONG __stdcall Release();
  private:
    long ref_count;
  };
```

L'implementazione dell'interfaccia *IInterface* è immediata:

```
ULONG __stdcall NotifierObject::AddRef()
{
  return InterlockedIncrement(&ref_count);
}

ULONG __stdcall NotifierObject::Release()
{
  ULONG result = InterlockedDecrement(&ref_count);
  if (ref_count == 0)
    delete this;
  return result;
}

HRESULT __stdcall NotifierObject::QueryInterface(const GUID& iid, void** obj)
{
  if (iid == __uuidof(IInterface)) {
    *obj = static_cast<IInterface*>(this);
    static_cast<IInterface*>(*obj)->AddRef();
    return S_OK;
  }
  if (iid == __uuidof(IOTANotifier)) {
    *obj = static_cast<IOTANotifier*>(this);
    static_cast<IOTANotifier*>(*obj)->AddRef();
    return S_OK;
  }
  return E_NOINTERFACE;
}
```

Benché i wizard ereditino da *IOTANotifier* e debbano implementare quindi tutte le sue funzioni, di solito l'IDE non fa uso di queste funzioni, e pertanto le implementazioni possono essere vuote:

```
void __fastcall NotifierObject::AfterSave()  {}
void __fastcall NotifierObject::BeforeSave() {}
void __fastcall NotifierObject::Destroyed()  {}
void __fastcall NotifierObject::Modified()   {}
```

Per utilizzare *NotifierObject* come classe base è necessario utilizzare l'eredità multipla. La classe wizard deve ereditare da *NotifierObject* e dalle interfacce wizard che devono essere implementate, come *IOTAWizard*. Poiché *IOTAWizard* eredita da *IOTANotifier* e da *IInterface*, nella classe derivata si crea ambiguità: le funzioni come *AddRef* () sono dichiarate in ogni ramo del grafico dell'albero genealogico dell'eredità. Per risolvere questo problema, scegliere una classe base come classe base principale e delegare tutte le funzioni ambigue a quella classe. Ad esempio, la dichiarazione della classe potrebbe apparire come segue:

```
class PACKAGE MyWizard : public NotifierObject, public IOTAMenuWizard {
```

```
    typedef NotifierObject inherited;
  public:
    // IOTAWizard
    virtual AnsiString __fastcall GetIDString();
    virtual AnsiString __fastcall GetName();
    virtual TWizardState __fastcall GetState();
    virtual void __fastcall Execute();

  // IOTAMenuWizard
    virtual AnsiString __fastcall GetMenuText();

    void __fastcall AfterSave();
    void __fastcall BeforeSave();
    void __fastcall Destroyed();
    void __fastcall Modified();
  protected:
    // IInterface
    virtual HRESULT __stdcall QueryInterface(const GUID&, void**);
    virtual ULONG __stdcall AddRef();
    virtual ULONG __stdcall Release();
  };
```

L'implementazione della classe potrebbe includere quanto segue:

```
ULONG __stdcall MyWizard::AddRef()  { return inherited::AddRef(); }
ULONG __stdcall MyWizard::Release() { return inherited::Release(); }
HRESULT __stdcall MyWizard::QueryInterface(const GUID& iid, void** obj)
{
  if (iid == __uuidof(IOTAMenuWizard)) {
    *obj = static_cast<IOTAMenuWizard*>(this);
    static_cast<IOTAMenuWizard*>(*obj)->AddRef();
    return S_OK;
  }
  if (iid == __uuidof(IOTAWizard)) {
    *obj = static_cast<IOTAWizard*>(this);
    static_cast<IOTAWizard*>(*obj)->AddRef();
    return S_OK;
  }
  return inherited::QueryInterface(iid, obj);
}
```

Poichè *AfterSave*, *BeforeSave*, e così via, hanno nella classe base corpi di funzione vuoti, nella classe derivata è possibile lasciarli come corpi di funzione vuoti ed evitare la chiamata non necessaria a *inherited::AfterSave* ().

Semplificazione dell'implementazione delle interfacce

Poiché le interfacce Object Pascal sono simili alle interfacce COM, è possibile utilizzare wizard COM come aiuto per la scrittura della classe wizard. Tuttavia, COM richiedono molto più lavoro rispetto a quello richiesto dalle semplici interfacce Object Pascal. Con un uso giudizioso di classi base e con qualche operazione di taglia e incolla, risulterà probabilmente più semplice non utilizzare i wizard COM, ma continuare ad utilizzare semplici implementazioni. Ad esempio, è possibile rendere più semplice l'implementazione di *QueryInterface* definendo una semplice macro:

```
#define QUERY_INTERFACE(T, iid, obj)    \
  if ((iid) == __uuidof(T)) {           \
    *(obj) = static_cast<T*>(this);     \
    static_cast<T*>(*(obj))->AddRef(); \
    return S_OK;                        \
  }
```

Utilizzare questa macro come segue:

```
HRESULT __stdcall MyWizard::QueryInterface(const GUID& iid, void* obj)
{
  QUERY_INTERFACE(IOTAMenuWizard, iid, obj);
  QUERY_INTERFACE(IOTAWizard, iid, obj);
  return inherited::QueryInterface(iid, obj);
}
```

È necessario anche ridefinire tutte le funzioni membro di *IOTAWizard* e le classi derivate che si utilizzano. La maggior parte delle funzioni delle diverse interfacce wizard è auto-esplicativa. L'IDE chiama le funzioni del wizard per decidere come presentare il wizard all'utente e come eseguire il wizard quando l'utente lo richiama.

Una volta terminata la scrittura della classe wizard, il passaggio successivo è l'installazione del wizard.

Installazione del package del wizard

Come per qualsiasi altro package di progettazione, un package di wizard deve avere almeno una funzione *Register*. (Vedere il Capitolo 52, "Come rendere disponibili i componenti in fase di progetto" per i dettagli sulla funzione *Register*.) Nella funzione *Register* è possibile registrare qualsiasi numero di wizard chiamando *RegisterPackageWizard* e passando un oggetto wizard come argomento unico, come verrà descritto in seguito:

```
namespace Example {
  void __fastcall PACKAGE Register()
  {
    RegisterPackageWizard(new MyWizard());
    RegisterPackageWizard(new MyOtherWizard());
  }
}
```

È possibile registrare anche editor di proprietà, componenti, e così via, come parti dello stesso package.

È bene ricordare che un package di progettazione è parte dell'applicazione principale di C++Builder, quindi qualsiasi nome di scheda deve essere univoco per l' intera applicazione e per tutti gli altri package di progettazione. Questo è lo svantaggio maggiore nell'utilizzo di package: non si possono conoscere i nomi che altri programmatori potrebbero attribuire alle loro schede.

Durante lo sviluppo, installare il package del wizard nel modo in cui si installerebbe qualsiasi altro package di progettazione: fare clic sul pulsante Install nel Package manager. L'IDE compilerà e collegherà il package e cercherà di caricarlo. L'IDE mostra una finestra di dialogo che informa se l'operazione è andata a buon fine.

Ottenimento dei servizi delle API Tools

Per fare qualcosa di utile, un wizard deve accedere all'IDE: ai suoi editor, alle sue finestre, al suo menu e così via. Questo è il ruolo delle interfacce di servizio. Le API Tools includono molti servizi, come i servizi delle azioni per eseguire le azioni dei file, i servizi dell'editor per accedere all'editor del codice sorgente, i servizi del debugger per accedere al debugger, e così via. La Tabella 58.2 riassume tutte le interfacce di servizio.

Tabella 58.2 Interfacce di servizi Tools API

Interfaccia	Descrizione
INTAServices	Consente l'accesso agli oggetti nativi IDE: menu principale, elenco delle azioni, lista delle immagini e barre strumenti.
IOTAActionServices	Esegue le azioni dei file di base: aprire, chiudere, salvare e caricare di nuovo un file.
IOTACodeCompletionServices	Consente l'accesso al completamento del codice, permettendo a un wizard di installare un manager custom per il completamento del codice.
IOTADebuggerServices	Consente l'accesso al debugger.
IOTAEditorServices	Consente l'accesso all' editor del codice sorgente e ai suoi buffer interni.
IOTAKeyBindingServices	Consente a un wizard di registrare connessioni di tastiera custom.
IOTAKeyboardServices	Consente l'accesso alle macro e alle connessioni di tastiera.
IOTAKeyboardDiagnostics	Attiva/disattiva il debug delle combinazioni di tasti.
IOTAMessageServices	Consente l'accesso alla visualizzazione dei messaggi.
IOTAModuleServices	Consente l'accesso per l'apertura dei file.
IOTAPackageServices	Richiede i nomi di tutti i package installati e dei loro componenti.
IOTAServices	Servizi vari.
IOTAToDoServices	Consente l'accesso alla lista To-do, permettendo a un wizard di installare un manager To-Do custom.
IOTAToolsFilter	Registra i notificatori di filtro strumenti.
IOTAWizardServices	Registra e annulla la registrazione dei wizard.

Per utilizzare un'interfaccia di servizio, convertire la variabile *BorlandIDEServices* nel servizio richiesto utilizzando la funzione membro Supports. Ad esempio,

```
void set_keystroke_debugging(bool debugging)
{
  _di_IOTAKeyboardDiagnostics diag;
  if (BorlandIDEServices->Supports(diag))
    diag->KeyTracing = debugging;
}
```

Se il wizard deve utilizzare spesso un servizio specifico, è possibile conservare un puntatore sul servizio come un membro dati della classe wizard. Utilizzando il modello *DelphiInterface* (per esempio, *_di_IOTAModuleServices*), le API Tools

gestiscono automaticamente il periodo di esistenza dell'oggetto e non è necessario fare niente di particolare nel distruttore del wizard.

Utilizzo di oggetti nativi dell'IDE

I wizard hanno completo accesso al menu principale, alle barre degli strumenti, all'elenco delle azioni e delle immagini dell'IDE. (si noti che molti menu contestuali dell'IDE non sono accessibili dalle API Tools). Questa sezione presenta un esempio semplice di come un wizard può utilizzare questi oggetti nativi dell'IDE per interagire con l'IDE.

Utilizzo dell'interfaccia INTAServices

L'interfaccia INTAServices è il punto di partenza per operare con oggetti nativi dell'IDE. Utilizzare questa interfaccia per aggiungere un'immagine alla lista delle immagini, un'azione alla lista delle azioni, una voce di menu al menu principale e un pulsante a una barra strumenti. È possibile legare l'azione alla voce di menu e al pulsante strumento. Quando il wizard viene distrutto, deve eliminare gli oggetti creati, ma non deve cancellare l'immagine che ha aggiunto alla lista di immagini. La cancellazione di un'immagine creerebbe disordine negli indici di tutte le immagini aggiunte dopo quel wizard.

Il wizard utilizza gli oggetti *TMainMenu*, *TActionList*, *TImageList* e *TToolBar* effettivi dell'IDE, ed è quindi possibile scrivere il codice nello stesso modo in cui si scriverebbero altre applicazioni. Inoltre ciò significa che potrebbe essere molto facile bloccare l'IDE o anche disattivare funzionalità importanti, come ad esempio cancellare il menu File.

Aggiunta di un'immagine alla lista delle immagini

Si supponga di volere aggiungere una voce di menu per richiamare il wizard. Si supponga inoltre di voler permettere all'utente di aggiungere un pulsante nella barra strumenti che richiami il wizard. Il primo passo è aggiungere un'immagine alla lista delle immagini dell'IDE. L'indice dell'immagine può essere quindi utilizzato per l'azione, che a sua volta, viene utilizzata dalla voce di menu e dal pulsante della barra strumenti. Utilizzare l'Image editor per creare un file di risorse che contenga una risorsa bitmap 16 per 16. Aggiungere il seguente codice al costruttore del wizard:

```
_di_INTAServices services;
BorlandIDEServices->Supports(services);

// Add an image to the image list.
Graphics::TBitmap* bitmap(new Graphics::TBitmap());
bitmap->LoadFromResourceName(reinterpret_cast<unsigned>(HInstance), "Bitmap1");
int image = services->AddMasked(bitmap, bitmap->TransparentColor,
                                "Tempest Software.intro wizard image");
delete bitmap;
```

Si noti che il tipo di *HInstance* è sbagliato e pertanto è necessario convertirlo nel tipo richiesto *daLoadFromResourceID*. Assicurarsi inoltre di caricare la risorsa tramite il nome o tramite l'ID specificato nel file di risorse. Aggiungere il file di risorse al package utilizzando la risorsa #pragma. È necessario scegliere un colore che sarà

interpretato come colore di sfondo per l'immagine. Se non si desidera un colore di sfondo, scegliere un colore che non esiste nella bitmap.

Aggiunta di un'azione all'elenco di azioni

L'indice delle immagini è utilizzato per creare un'azione, come viene mostrato in seguito. Il wizard utilizza gli eventi *OnExecute* e *OnUpdate*. Uno scenario comune è che un wizard utilizzi l'evento *OnUpdate* per attivare o disattivare l'azione. Assicurarsi che l'evento *OnUpdate* ritorni velocemente, altrimenti si noterà che l'IDE diventa lenta dopo avere caricato il wizard. L'evento *OnExecute* dell'azione è simile al metodo *Execute* del wizard. Se si utilizza una voce di menu per richiamare un wizard di scheda o di progetto, si potrebbe anche volere fare chiamare direttamente *Execute* da *OnExecute*.

```
action = new TAction(0);
action->ActionList = services->ActionList;
action->Caption    = GetMenuText();
action->Hint       = "Display a silly dialog box";
action->ImageIndex = image;
action->OnUpdate   = action_update;
action->OnExecute  = action_execute;
```

La voce di menu imposta la sua proprietà *Action* all'azione appena creata. L'astuzia nel creare una voce di menu sta nel sapere dove inserirla. L'esempio che segue cerca il menu View e inserisce la nuova voce di menu come prima voce nel menu View. (Di solito, contare su una posizione assoluta non è una buona idea): non si sa mai quando un altro wizard si potrebbe inserire nel menu. È probabile che le versioni future di C++Builder riordineranno il menu. Un metodo migliore è esaminare il menu cercando una voce di menu con un nome specifico. Il metodo semplificato viene mostrato di seguito per chiarezza).

```
for (int i = 0; i < services->MainMenu->Items->Count; ++i)
{
  TMenuItem* item = services->MainMenu->Items->Items[i];
  if (CompareText(item->Name, "ViewsMenu") == 0)
  {
    menu_item = new TMenuItem(0);
    menu_item->Action = action;
    item->Insert(0, menu_item);
  }
}
```

Aggiungendo l'azione all'elenco delle azioni dell'IDE, l'utente può vedere l'azione mentre personalizza le barre strumenti. L'utente può selezionare l'azione e aggiungerla come pulsante a una qualsiasi barra strumenti. Ciò può causare un problema quando viene scaricato il wizard: tutti i pulsanti strumento finiscono per dirottare i puntatori sull'azione inesistente e sul gestore di evento *OnClicK*. Per evitare violazioni di accesso, il wizard deve trovare tutti i pulsanti strumento che fanno riferimento alla sua azione e rimuoverli.

Cancellazione di pulsanti dalla barra strumenti

Non esiste una funzione pratica per la rimozione di un pulsante da una barra strumenti; è necessario inviare il messaggio CM_CONTROLCHANGE, in cui il

primo parametro è il controllo da cambiare e il secondo parametro è zero se si desidera rimuovere quel controllo oppure diverso da zero se si desidera aggiungerlo alla barra degli strumenti. Dopo avere rimosso i pulsanti dalla barra degli strumenti, il distruttore cancella l'azione e la voce di menu. La cancellazione di questi elementi provoca automaticamente la loro rimozione dall'ActionList e dal MainMenu dell'IDE.

```
void __fastcall remove_action (TAction* action, TToolBar* toolbar)
{
  for (int i = toolbar->ButtonCount; --i >= 0; )
  {
    TToolButton* button = toolbar->Buttons[i];
    if (button->Action == action)
    {
      // Remove "button" from "toolbar".
      toolbar->Perform(CM_CONTROLCHANGE, WPARAM(button), 0);
      delete button;
    }
  }
}

__fastcall MyWizard::~MyWizard()
{
  _di_INTAServices services;
  BorlandIDEServices->Supports(services);
  // Check all the toolbars, and remove any buttons that use
  // this action.
  remove_action(action, services->ToolBar[sCustomToolBar]);
  remove_action(action, services->ToolBar[sDesktopToolBar]);
  remove_action(action, services->ToolBar[sStandardToolBar]);
  remove_action(action, services->ToolBar[sDebugToolBar]);
  remove_action(action, services->ToolBar[sViewToolBar]);
  remove_action(action, services->ToolBar[sInternetToolBar]);

  delete menu_item;
  delete action;
}
```

Come si può notare da questo semplice esempio, il wizard interagisce con l'IDE in modo molto flessibile. Tuttavia, questa flessibilità richiede maggiore responsabilità. È facile incorrere in puntatori non corretti o compiere altre violazioni di accesso. Nella sezione successiva vengono riportati alcuni suggerimenti su come diagnosticare questo genere di errori.

Debug di un wizard

Quando si scrivono wizard che utilizzano le API tools native. può succedere di scrivere del codice che provoca il blocco dell'IDE. Può inoltre succedere di scrivere un wizard che si installa ma che non si comporta nel modo desiderato. Una delle sfide maggiori quando si opera con del codice da utilizzare in fase di progettazione è rappresentata dalle operazioni di debug. Tuttavia, è un problema che si può risolvere facilmente. Poiché il wizard è installato in C++Builder stesso, è sufficiente solo

impostare, mediante il comando di menu Run|Parameters..., l'Host Application del package all'eseguibile di C++Builder (bcb.exe).

Quando si desidera (o si deve) eseguire il debug del package, non lo si deve installare. Scegliere invece Run|Run dalla barra dei menu. Ciò avvia una nuova istanza di C++Builder. Nella nuova istanza, installare il package già compilato scegliendo Components|Install Package... dalla barra dei menu. Nell'istanza originaria di C++Builder, si dovrebbero vedere ora i punti blu rilevatori che indicano dove è possibile collocare i breakpoint nel codice sorgente del wizard. (Nel caso non si vedano, ricontrollare le opzioni del compilatore per assicurarsi di avere attivato il debug; assicurarsi di avere caricato il package corretto; ricontrollare i moduli di processo per essere assolutamente certi di avere caricato il file .bpl desiderato).

Con questa procedura non è possibile eseguire il debug nella VCL, nella CLX o nel codice RTL, ma si hanno tutte le possibilità di eseguire il debug del wizard stesso, il che potrebbe essere sufficiente per individuare cosa non funziona.

Numeri di versione dell'interfaccia

Se si osservano attentamente le dichiarazioni di alcune interfacce, come *IOTAMessageServices*, si vedrà che esse ereditano da altre interfacce con nomi simili, come *IOTAMessageServices50*, che eredita a sua volta da *IOTAMessageServices40*. L'utilizzo di numeri di versione aiuta a isolare il codice da cambiamenti tra le varie versioni di C++Builder.

Le API Tools seguono il principio di base di COM secondo il quale un'interfaccia e il suo GUID non cambiano mai. Se una nuova versione aggiunge delle funzioni a un'interfaccia, le API Tools dichiarano una nuova interfaccia che eredita da quella precedente. Il GUID rimane uguale, collegato all'interfaccia precedente invariata. La nuova interfaccia ottiene un GUID completamente nuovo. I wizard precedenti che utilizzano il GUID precedente continuano a funzionare.

Le API Tools cambiano anche i nomi dell'interfaccia per provare a salvaguardare la compatibilità del codice sorgente. Per vedere come ciò funziona, è importante distinguere fra i due tipi di interfacce nelle API Tools: quelle implementate da Borland e quelle implementate dall'utente. Se l'IDE implementa l'interfaccia, il nome corrisponde alla versione più recente dell'interfaccia. La nuova funzionalità non influisce sul codice esistente. Alle vecchie interfacce viene aggiunto il numero della versione precedente.

Per un'interfaccia implementata dall'utente, tuttavia, nuove funzioni membro nell'interfaccia di base richiedono nuove funzioni nel codice. Quindi, il nome tende a corrispondere alla vecchia interfaccia e alla nuova interfaccia viene aggiunto un numero di versione alla fine.

Si considerino ad esempio i servizi dei messaggi. C++Builder 6 ha introdotto una nuova funzione: gruppi di messaggi. Quindi, l'interfaccia dei servizi base per i messaggi ha richiesto nuove funzioni membro. Queste funzioni sono state dichiarate in una nuova classe di interfaccia, che ha mantenuto il nome *IOTAMessageServices*. La vecchia interfaccia per i servizi di messaggi è stata rinominata *IOTAMessageServices50*

(per la versione 5). Il GUID del vecchio *IOTAMessageServices* è lo stesso di quello del nuovo *IOTAMessageServices50* poiché le funzioni membro sono le stesse.

Si consideri *IOTAIDENotifier* come esempio di interfaccia implementata dall'utente. C++Builder 5 ha aggiunto nuove funzioni in overload: *AfterCompile* e *BeforeCompile*. Non è stato necessario cambiare il codice esistente utilizzava *IOTAIDENotifier*, ma il nuovo codice che richiedeva la nuova funzionalità doveva essere modificato per ridefinire le nuove funzioni ereditate da *IOTAIDENotifier50*. La versione 6 non ha aggiunto ulteriori funzioni, pertanto la versione corrente da utilizzare è *IOTAIDENotifier50*.

La regola empirica quando si scrive del nuovo codice è utilizzare la classe più derivata. Ignorare il codice sorgente se si ricompila semplicemente un wizard esistente con una nuova versione di C++Builder.

Operazioni con file ed editor

Prima di proseguire, è necessario capire come le API Tools operano con i file. L'interfaccia principale è *IOTAModule*. Un modulo rappresenta un insieme di file aperti logicamente collegati. Ad esempio, un singolo modulo rappresenta una singola unit. Il modulo, a sua volta, ha uno o più editor, e ogni editor rappresenta un file, ad esempio i file di implementazione (.cpp), di interfaccia (.h) o di scheda (.dfm o .xfm). Le interfacce dell'editor riflettono lo stato interno degli editor dell'IDE, e pertanto un wizard può vedere il codice e le schede modificati che l'utente vede, anche se l'utente non ha salvato nessun cambiamento.

Utilizzo delle interfacce di modulo

Per ottenere un'interfaccia di modulo, iniziare con i servizi di modulo (*IOTAModuleServices*). È possibile interrogare i servizi di modulo di tutti i moduli aperti, cercare un modulo in base a un nome di archivio o di scheda o aprire un file per ottenere la sua interfaccia di modulo.

Ci sono vari tipi di moduli per vari tipi di file, come progetti, risorse e librerie di tipi. Convertire un interfaccia di modulo a un genere specifico di interfaccia di modulo per sapere se il modulo è di quel tipo. Ad esempio, il seguente è un modo per ottenere l'interfaccia corrente di gruppo del progetto:

```
// Return the current project group, or 0 if there is no project group.
_di_IOTAProjectGroup __fastcall CurrentProjectGroup()
{
  _di_IOTAModuleServices svc;
  BorlandIDEServices->Supports(svc);

  for (int i = 0; i < svc->ModuleCount; ++i)
  {
    _di_IOTAModule module = svc->Modules[i];
    _di_IOTAProjectGroup group;
    if (module->Supports(group))
      return group;
  }
```

```
    return 0;
  }
```

Utilizzo delle interfacce degli editor

Ogni modulo ha almeno un interfaccia di editor. Alcuni moduli hanno più editor, come un file di implementazione (.cpp), un file di interfaccia (.h) e un file di descrizione di scheda(.dfm). Tutti gli editor implementano l'interfaccia *IOTAEditor* ; convertono l'editor in un tipo specifico per sapere di che tipo di editor si tratta. Ad esempio, per ottenere l'interfaccia dell'editor di scheda per una unit, è possibile fare quanto segue:

```
// Return the form editor for a module, or 0 if the unit has no form.
_di_IOTAFormEditor __fastcall GetFormEditor(_di_IOTAModule module)
{
  for (int i = 0; i < module->ModuleFileCount; ++i)
  {
    _di_IOTAEditor editor = module->ModuleFileEditors[i];
    _di_IOTAFormEditor formEditor;
    if (editor->Supports(formEditor))
      return formEditor;
  }
  return 0;
}
```

Le interfacce degli editor danno accesso al loro stato interno. È possibile esaminare il codice sorgente o i componenti che l'utente sta modificando, apportare modifiche al codice sorgente, ai componenti o alle proprietà, modificare la selezione nell'editor sorgente e dell'editor delle schede ed eseguire quasi tutte le azioni dell'editor che possono essere eseguite dall'utente.

Un wizard può accedere a tutti i componenti della scheda utilizzando un'interfaccia dell'editor della scheda. Ogni componente (tra cui la scheda principale o il modulo dati) ha un'interfaccia associata *IOTAComponent*. Un wizard può esaminare o cambiare la maggior parte delle proprietà dei componente. Se è necessario avere controllo assoluto sul componente, è possibile convertire l'interfaccia *IOTAComponent* in *INTAComponent*. L'interfaccia nativa del componente consente al wizard di accedere direttamente al puntatore *INTAComponent*. Questo potrebbe essere importante se fosse necessario leggere o modificare una proprietà di tipo classe, come *TFont*, il che è possibile solo tramite interfacce in stile NTA.

Creazione di schede e di progetti

C++Builder dispone già di molti wizard per schede e progetti, ed è possibile scriverne di propri. L'Object Repository permette di creare modelli statici che possono essere utilizzati in un progetto, ma un wizard è molto più potente perchè è dinamico. Il wizard può interagire con l'utente e creare vari tipi di file a seconda delle risposte dell'utente. Questa sezione descrive come scrivere un wizard per scheda o per progetto.

Creazione dei moduli

Di solito un wizard per scheda o per progetto crea uno o più file nuovi. Tuttavia al posto di file reali, sarebbe meglio creare moduli senza nome e non salvati. Quando l'utente li salva, l'IDE richiede all'utente un nome per il file. Un wizard utilizza un oggetto creator per creare tali moduli.

Una classe creator implementa un'interfaccia creator, la quale eredita da *IOTACreator*. Il wizard passa un oggetto creator al metodo *CreateModule* del servizio del modulo e l'IDE richiama l'oggetto creator per ottenere i parametri di cui ha bisogno per creare il modulo.

Ad esempio, un wizard per scheda che crea una nuova scheda implementa di solito *GetExisting()* per restituire **false** e *GetUnnamed()* per restituire **true**. Ciò crea un modulo che non ha nome (in modo che l'utente debba scegliere un nome prima che il file possa essere salvato) e non si appoggia ad alcun file esistente (in modo che l'utente debba salvare il file anche se non viene apportato nessun cambiamento). Altri metodi del creator comunicano all'IDE il tipo di file che viene creato (per esempio, progetto, unit o scheda), forniscono il contenuto del file o restituiscono il nome della scheda, il nome dell'antenato e altre importanti informazioni. Ulteriori callback consentono a un wizard di aggiungere moduli a un progetto appena creato o aggiungono componenti a una scheda appena creata.

Per creare un nuovo file, come viene spesso richiesto in un wizard per scheda o per progetto, è di solito necessario fornirne i contenuti. Per fare ciò, scrivere una nuova classe che implementa l'interfaccia *IOTAFile*. Se il wizard è in grado di farlo con i contenuti predefiniti del file, è possibile restituire un puntatore 0 da qualsiasi funzione che restituisce *IOTAFile*.

Ad esempio, si supponga che l'azienda per cui si lavora abbia un blocco di commento standard che deve apparire all'inizio di ogni file sorgente. Si potrebbe fare ciò utilizzando un modello statico nell'Object Repository, ma potrebbe essere necessario aggiornare manualmente il blocco di commento per riportare l'autore e la data di creazione. Invece, è possibile utilizzare un creator per riempire in modo dinamico il blocco di commento al momento della creazione del file.

Il primo passo consiste nello scrivere un wizard che crea nuove unit e nuove schede. La maggior parte delle funzioni del creator restituiscono zero, stringhe vuote o altri valori predefiniti, i quali comunicano alle API Tools di creare una nuova unit o una nuova scheda con il comportamento predefinito. Ridefinire *GetCreatorType* per comunicare alle API Tools se devono creare una unit oppure una scheda. Per creare una unit, restituire la macro sUnit. Per creare una scheda, restituire sForm. Per semplificare il codice, utilizzare una singola classe che accetti come argomento del costruttore il tipo creator. Salvare il tipo creator in un membro dati, in modo da permettere a *GetCreatorType* di restituirne il valore. Ridefinire *NewImplSource* e *NewIntfSource* per restituire i contenuti del file desiderati.

```
class PACKAGE Creator : public IOTAModuleCreator {
public:
  __fastcall Creator(const AnsiString creator_type)
  : ref_count(0), creator_type(creator_type) {}
  virtual __fastcall ~Creator();
```

```
// IOTAModuleCreator
  virtual AnsiString __fastcall GetAncestorName();
  virtual AnsiString __fastcall GetImplFileName();
  virtual AnsiString __fastcall GetIntfFileName();
  virtual AnsiString __fastcall GetFormName();
  virtual bool __fastcall  GetMainForm();
  virtual bool __fastcall  GetShowForm();
  virtual bool __fastcall  GetShowSource();
  virtual _di_IOTAFile __fastcall NewFormFile(
    const AnsiString FormIdent, const AnsiString AncestorIdent);
  virtual _di_IOTAFile __fastcall NewImplSource(
    const AnsiString ModuleIdent, const AnsiString FormIdent,
    const AnsiString AncestorIdent);
  virtual _di_IOTAFile __fastcall NewIntfSource(
    const AnsiString ModuleIdent, const AnsiString FormIdent,
    const AnsiString AncestorIdent);
  virtual void __fastcall  FormCreated(
    const _di_IOTAFormEditor FormEditor);

// IOTACreator
  virtual AnsiString __fastcall GetCreatorType();
  virtual bool __fastcall  GetExisting();
  virtual AnsiString __fastcall GetFileSystem();
  virtual _di_IOTAModule __fastcall GetOwner();
  virtual bool __fastcall  GetUnnamed();

protected:
  // IInterface
  virtual HRESULT __stdcall QueryInterface(const GUID&, void**);
  virtual ULONG __stdcall AddRef();
  virtual ULONG __stdcall Release();

private:
  long ref_count;
  const AnsiString creator_type;
};
```

La maggior parte dei membri di *Creator* restituiscono zero o stringhe vuote. I metodi booleani restituiscono **true**, tranne *GetExisting*, che restituisce **false**. Il metodo più interessante è *GetOwner*, che restituisce un puntatore al modulo del progetto corrente, oppure 0 se non esiste alcun progetto. Non esiste un modo semplice per scoprire il progetto corrente o il gruppo di progetti corrente. *GetOwner* deve esaminare in sequenza invece tutti i moduli aperti. Se viene trovato un gruppo di progetti, deve essere l'unico gruppo di progetti aperto, in modo che *GetOwner* restituisca il suo progetto corrente. Altrimenti, la funzione restituisce il primo modulo di progetto che trova, oppure 0 se non è aperto alcun progetto.

```
_di_IOTAModule __fastcall Creator::GetOwner()
{
  // Return the current project.
  _di_IOTAProject result = 0;

  _di_IOTAModuleServices svc = interface_cast<IOTAModuleServices>(BorlandIDEServices);
  for (int i = 0; i < svc->ModuleCount; ++i)
  begin
    _di_IOTAModule module = svc->Modules[i];
```

```
      _di_IOTAProject project;
      _di_IOTAProjectGroup group;
      if (module->Supports(project)) {
        // Remember the first project module.
        if (result == 0)
          result = project;
      } else if (module->Supports(group)) {
        // Found the project group, so return its active project.
        result = group->ActiveProject;
        break;
      }
    }
    return result;
  }
```

Il creator restituisce 0 da *NewFormSource*, per generare un file scheda predefinito. I metodi interessanti sono *NewImplSource* e *NewIntfSource*, che creano un'istanza di *IOTAFile* che restituisce i contenuti del file.

La classe *File* implementa l'interfaccia *IOTAFile*. Questa interfaccia restituisce –1 come età del file (il che significa che il file non esiste) e restituisce i contenuti del file come stringa. Per mantenere semplice la classe *File*, il creator genera la stringa e la classe *File* la trasmette semplicemente.

```
class File : public IOTAFile {
public:
  __fastcall File(const AnsiString source);
  virtual __fastcall ~File();
  AnsiString __fastcall GetSource();
  System::TDateTime __fastcall GetAge();
protected:
  // IInterface
  virtual HRESULT __stdcall QueryInterface(const GUID&, void**);
  virtual ULONG __stdcall AddRef();
  virtual ULONG __stdcall Release();
private:
  long ref_count;
  AnsiString source;
};

__fastcall File::File(const AnsiString source)
: ref_count(0), source(source)
{}

AnsiString __fastcall File::GetSource()
{
  return source;
}

System::TDateTime __fastcall File::GetAge()
{
  return -1;
}
```

È possibile conservare il testo del contenuto del file in una risorsa per renderne più semplice la modifica, ma per maggiore semplicità, questo esempio scrive

direttamente il codice sorgente nel wizard. Supponendo che ci sia una scheda, l'esempio seguente genera il codice sorgente. È possibile aggiungere con facilità il caso più semplice di una unit normale. Controllare *FormIdent* e se è vuoto, creare una unit normale; altrimenti creare una unit di scheda. Lo scheletro di base del codice è lo stesso dell'impostazione predefinita dell'IDE (con l'aggiunta iniziale dei commenti, naturalmente), ma è possibile modificarlo nel modo desiderato.

```
_di_IOTAFile __fastcall Creator::NewImplSource(
  const AnsiString ModuleIdent,
  const AnsiString FormIdent,
  const AnsiString AncestorIdent)
{
  const AnsiString form_source =
"/*-----------------------------------------------------------------\n"
" %m - description\n"
" Copyright © %y Your company, inc.\n"
" Created on %d\n"
" By %u\n"
"  ---------------------------------------------------------------*/\n"
"\n"
"#include <vcl.h>\n"
"#pragma hdrstop\n"
"\n"
"#include \"%m.h\"\n"
"//-----------------------------------------------------------------\n"
"#pragma package(smart_init)\n"
"#pragma resource \"*.dfm\"\n"
"T%f *%f;\n"
"//-----------------------------------------------------------------\n"
"__fastcall T%m::T%m(TComponent* Owner)\n"
"        : T%a(Owner)\n"
"{\n"
"}\n"
"//----------------------------------------------------------------\n";

  return new File(expand(form_source, ModuleIdent, FormIdent,
                       AncestorIdent));
}
```

Si noti che il codice sorgente contiene stringhe con formato %m e %y. Questi sono concettualmente simili a printf o ai controlli Format, ma vengono espansi dalla funzione di espansione del wizard. Più precisamente, %m viene espanso nell'identificativo del modulo o della unit, %f nel nome della scheda e %a nel nome dell'antenato. Si noti come il nome della schada vengo utilizzato per il nome del tipo della scheda con l'aggiunta di una T maiuscola. Alcuni specificatori di formato aggiuntivi semplificano la generazione del blocco di commento: %d per la data, %u per l'utente e %y per l'anno. (La scrittura della funzione di espansione è indipendente dalle API Tools ed è proposto come un esercizio per il lettore.)

NewIntfSource è simile a *NewImplSource*, ma genera il file dell'interfaccia (.h).

Il passo finale consiste nella creazione di due wizard per scheda: uno utilizza sUnit come tipo di creator e l'altro utilizza sForm. Con ulteriore vantaggio per l'utente, è possibile utilizzare *INTAServices* per aggiungere una voce di menu al menu File|

New al fine di richiamare ciascun wizard. Il gestore di evento *OnClick* della voce di menu può chiamare la funzione *Execute* del wizard.

Alcuni wizard possono avere la necessità di attivare o disattivare le voci di menu, a seconda di ciò che sta avvenendo nell'IDE. Ad esempio, un wizard che controlla un progetto in un sistema di controllo del codice sorgente dovrebbe disattivare la sua voce di menu Check In se nell'IDE non è aperto alcun file. È possibile aggiungere questa funzione al wizard utilizzando i notificatori, come viene spiegato nella sezione successiva.

Notifica a un wizard di eventi dell'IDE

Un aspetto importante della scrittura di un wizard con un comportamento corretto consiste nel fare in modo che il wizard risponda agli eventi dell'IDE. In particolare, qualsiasi wizard che tenga traccia delle interfacce dei moduli deve essere in grado di rilevare quando l'utente chiude il modulo, in modo che il wizard possa rilasciare l'interfaccia. Per fare ciò, il wizard necessita di un notificatore, il che significa che è necessario scrivere una classe notifier.

Tutte le classi notifier implementano una o più interfacce notifier. Le interfacce notifier definiscono i metodi delle callback; il wizard registra un oggetto notifier con le API Tools e l'IDE richiama il notificatore quando accade qualcosa di importante.

Ogni interfaccia notifier eredita da *IOTANotifier*, benché non tutti i suoi metodi vengano utilizzati per un particolare notificatore. LaTabella 58.3 elenca tutte le interfacce notifier e fornisce una breve descrizione per ognuna di esse.

Tabella 58.3 Le interfacce notifier

Interfaccia	Descrizione
IOTANotifier	Classe base astrattaper tutti i notificatori
IOTABreakpointNotifier	Per innescare o sostituire un breakpoint nel debugger
IOTADebuggerNotifier	Per eseguire un programma nel debugger o per aggiungere o cancellare breakpoint
IOTAEditLineNotifier	Per tracciare gli spostamenti delle righe nell'editor dei sorgenti
IOTAEditorNotifier	Per modificare o salvare un file sorgente o cambiare i file nell'editor
IOTAFormNotifier	Per salvare una scheda oppure modificare la scheda o qualsiasi suo componente (o modulo dati)
IOTAIDENotifier	Per caricare progetti, per installare package e per altri eventi globali dell'IDE
IOTAMessageNotifier	Per aggiungere o rimuovere pagine (gruppi di messaggio) nella vista dei messaggi
IOTAModuleNotifier	Per modificare, salvare o rinominare un modulo
IOTAProcessModNotifier	Per caricare un modulo di processo nel debugger
IOTAProcessNotifier	Per creare o distruggere thread e processi nel debugger
IOTAThreadNotifier	Per modificare lo stato di un thread nel debugger
IOTAToolsFilterNotifier	Per richiamare un filtro strumenti

Per vedere come utilizzare i notificatori, si consideri l'esempio precedente. Mediante i creator di modulo, l'esempio crea un wizard che aggiunge un commento ad ogni file sorgente. Il commento include il nome iniziale della unit, ma l'utente salva quasi sempre il file con un nome diverso. In questo caso, sarebbe bello se il wizard aggiornasse il commento in modo da farlo corrispondere al vero nome del file.

Per fare ciò, è necessario un notificatore di modulo. Il wizard salva l'interfaccia del modulo che *CreateModule* restituisce e la utilizza per registrare un notificatore di modulo. Il notificatore di modulo riceve la notifica quando l'utente modifica il file oppure lo salva, ma questi eventi non sono importanti per questo wizard, e pertanto *AfterSave* e le relative funzioni hanno corpi vuoti. La funzione importante è *ModuleRenamed*, che l'IDE chiama quando l'utente salva il file con un nuovo nome. La dichiarazione per la classe notifier del modulo è la seguente:

```
class ModuleNotifier : public NotifierObject, public IOTAModuleNotifier
{
  typedef NotifierObject inherited;
public:
  __fastcall ModuleNotifier(const _di_IOTAModule module);
  __fastcall ~ModuleNotifier();

// IOTAModuleNotifier
  virtual bool __fastcall  CheckOverwrite();
  virtual void __fastcall  ModuleRenamed(const AnsiString NewName);

// IOTANotifier
  void __fastcall AfterSave();
  void __fastcall BeforeSave();
  void __fastcall Destroyed();
  void __fastcall Modified();
protected:
  // IInterface
  virtual HRESULT __stdcall QueryInterface(const GUID&, void**);
  virtual ULONG __stdcall AddRef();
  virtual ULONG __stdcall Release();
private:
  _di_IOTAModule module;
  AnsiString name;        // Remember the module's old name.
  int index;              // Notifier index.
};
```

Un modo per scrivere un notificatore consiste nel fare in modo che esso si registri automaticamente nel proprio costruttore. Il distruttore annulla la registrazione del notificatore. Nel caso di un notificatore di modulo, l'IDE chiama il metodo *Destroyed* quando l'utente chiude il file. In quel caso, il notificatore deve annullare la propria registrazione e rilasciare il suo riferimento all'interfaccia del modulo. L'IDE rilascia il proprio riferimento al notificatore, il che riporta a zero il contatore dei riferimenti e libera l'oggetto. Pertanto, è necessario scrivere il distruttore con molta attenzione: la registrazione del notificatore potrebbe già essere stata annullata.

```
__fastcall ModuleNotifier::ModuleNotifier(const _di_IOTAModule module)
: index(-1), module(module)
{
  // Register this notifier.
  index = module->AddNotifier(this);
```

```
  // Remember the module's old name.
  name = ChangeFileExt(ExtractFileName(module->FileName), "");
}

__fastcall ModuleNotifier::~ModuleNotifier()
{
  // Unregister the notifier if that hasn't happened already.
  if (index >= 0)
    module->RemoveNotifier(index);
}

void __fastcall ModuleNotifier::Destroyed()
{
  // The module interface is being destroyed, so clean up the notifier.
  if (index >= 0)
  {
    // Unregister the notifier.
    module->RemoveNotifier(index);
    index = -1;
  }
  module = 0;
}
```

L'IDE richiama la funzione *ModuleRenamed* del notificatore quando l'utente rinomina il file. La funzione accetta come parametro il nuovo nome, che il wizard utilizza per aggiornare il commento nel file. Per editare il buffer del sorgente, il wizard utilizza un'interfaccia edit position. Il wizard trova la posizione giusta, ricontrolla di avere trovato il testo giusto e sostituisce quel testo con il nuovo nome.

```
void __fastcall ModuleNotifier::ModuleRenamed(const AnsiString NewName)
{
  // Get the module name from the new file name.
  AnsiString ModuleName = ChangeFileExt(ExtractFileName(NewName), "");
  for (int i = 0; i < module->GetModuleFileCount(); ++i)
  {
    // Update every source editor buffer.
    _di_IOTAEditor editor = module->GetModuleFileEditor(i);
    _di_IOTAEditBuffer buffer;
    if (editor->Supports(buffer))
    {
      _di_IOTAEditPosition pos = buffer->GetEditPosition();
      // The module name is on line 2 of the comment.
      // Skip leading white space and copy the old module name,
      // to double check we have the right spot.
      pos->Move(2, 1);
      pos->MoveCursor(mmSkipWhite | mmSkipRight);
      AnsiString check = pos->RipText("", rfIncludeNumericChars | rfIncludeAlphaChars);
      if (check == name)
      {
        pos->Delete(check.Length());    // Delete the old name.
        pos->InsertText(ModuleName);    // Insert the new name.
        name = ModuleName;              // Remember the new name.
      }
    }
  }
}
```

Cosa accade se l'utente inserisce degli altri commenti prima del nome del modulo? In questo caso, è necessario utilizzare un notificatore edit line per tenere traccia del numero di riga in cui si trova si trova il nome del modulo. Per fare ciò, utilizzare le interfacce *IOTAEditLineNotifier* e *IOTAEditLineTracker*, descritte nella guida in linea.

È bene essere prudenti durante la scrittura dei notificatori. È necessario assicurarsi che nessun notificatore sopravviva al suo wizard. Ad esempio, se l'utente utilizzasse il wizard per creare una nuova unit e quindi scaricasse il wizard, ci sarebbe ancora un notificatore collegato alla unit. I risultati sarebbero imprevedibili, ma molto probabilmente, l'IDE si bloccherebbe. Quindi, il wizard deve tenere traccia di tutti i suoi notificatori e deve annullare la registrazione di ogni notificatore prima che il wizard venga distrutto. D'altra parte, se l'utente per prima cosa chiudesse il file, il notificatore di modulo riceverebbe una notifica *Destroyed*, il che significa che il notificatore deve annullare la propria registrazione e rilasciare tutti i riferimenti al modulo. Il notificatore si deve rimuovere anche dall'elenco principale dei notificatori del wizard.

Di seguito viene riportata la versione finale della funzione *Execute* del wizard. Questa funzione crea il nuovo modulo, utilizza l'interfaccia del modulo e crea un notificatore di modulo, quindi lo salva in un elenco di interfacce (*TInterfaceList*).

```
void __fastcall DocWizard::Execute()
{
  _di_IOTAModuleServices svc;
  BorlandIDEServices->Supports(svc);
  _di_IOTAModule module = svc->CreateModule(new Creator(creator_type));
  _di_IOTAModuleNotifier notifier = new ModuleNotifier(module);
  list->Add(notifier);
}
```

Il distruttore del wizard itera passa in esame l'elenco delle interfacce e annulla la registrazione di ogni notificatore nell'elenco. Il permettere semplicemente all'elenco delle interfacce di rilasciare le interfacce elencate non è sufficiente poiché anche l'IDE conserva le stesse interfacce. È necessario comunicare all'IDE di rilasciare le interfacce notifier per poter liberare gli oggetti notifier. In questo caso, il distruttore inganna i notificatori facendo credere loro che i rispettivi moduli siano stati distrutti. In una situazione più complicata, si trarrebbe maggior vantaggio nello scrivere una funzione Unregister separata per la classe notifier.

```
__fastcall DocWizard::~DocWizard()
{
  // Unregister all the notifiers in the list.
  for (int i = list->Count; --i >= 0; )
  {
    _di_IOTANotifier notifier;
    list->Items[i]->Supports(notifier);
    // Pretend the associated object has been destroyed.
    // That convinces the notifier to clean itself up.
    notifier->Destroyed();
    list->Delete(i);
  }
  delete list;
  delete item;
}
```

Il resto del wizard gestisce i dettagli di routine della registrazione del wizard, installando voci di menu e cose simili. La sezione successiva propone ulteriori dettagli sulla registrazione di un wizard illustrando come eseguire l'operazione con una DLL invece che con un package.

Installazione della DLL di un wizard

È possibile installare un wizard anche con una DLL. Assicurarsi che il wizard utilizzi la RTL dinamica e i package di esecuzione. Aggiungere il package designide all'elenco dei package di esecuzione in Project Options. La DLL deve esportare una funzione di inizializzazione il cui nome è INITWIZARD0001. Quando l'IDE carica la DLL, cerca il nome di esportazione speciale e chiama quella funzione. Per esportare la funzione è possibile utilizzare un file di definizione del modulo oppure utilizzare una dichiarazione __declspec(dllexport). In questo caso, è necessario anche evitare di modificare il nome; dichiarare poi la funzione come extern "C".

La funzione deve essere di tipo *TWizardInitProc*. Essa accetta come uno degli argomenti un puntatore a una funzione di registrazione. Essa chiama questa funzione per registrare ogni oggetto wizard, nello stesso modo in cui un wizard di package chiama *RegisterPackageWizard*. La funzione di inizializzazione dovrebbe restituire **true** nel caso l'operazione sia andata a buon fine oppure **false** in caso contrario. Il seguente esempio mostra come scrivere la funzione di inizializzazione.

```
extern "C" bool __stdcall __declspec (dllexport) INITWIZARD0001(
  const _di_IBorlandIDEServices,
  TWizardRegisterProc RegisterProc,
  TWizardTerminateProc&)
{
  RegisterProc(new MyWizard());
  RegisterProc(new MyOtherWizard());
  return true;
}
```

Se si utilizzano package di esecuzione, il primo argomento per la funzione di inizializzazione non è importante. (Il primo parametro è trattato in "Utilizzo di una DLL senza package di esecuzione" a pagina 58-24) L'argomento finale è un riferimento a un puntatore di funzione di una procedura di terminazione. È possibile utilizzare questa funzione per eseguire una pulizia globale. Di solito, è possibile ignorare l'ultimo parametro perché il wizard dovrebbe pulirsi da sé nel suo distruttore.

Per installare la DLL, aggiungere un elemento al file di registro con la seguente chiave:

```
HKEY_CURRENT_USER\Software\Borland\C++Builder\6.0\Experts
```

Il nome dell'elemento dovrebbe essere un nome univoco, come la stringa ID del wizard. Il valore dovrebbe essere il percorso completo per la DLL. Al successio avvio, C++Builder controlla il file di registro e carica tutte le DLL elencate sotto la chiave Experts. La DLL rimane caricata mentre l'IDE è in esecuzione. Quando C++Builder termina, scarica la DLL. Ciò rallenta il debug, che è il motivo per cui durante lo

sviluppo è bene preferire i package di progettazione. Quando si è pronti a rilasciare il wizard, è possibile trasformare un package di progettazione in una DLL.

Se C++Builder comunica che la versione del wizard è sbagliata, significa che non è in grado di trovare la funzione INITWIZARD0001. Rincontrollare di avere scritto il nome correttamente e di averlo esportato senza modifiche.

Il vantaggio principale dell'utilizzo di DLL invece che di package consiste nel fatto che si evita il problema della collisione dei nomi. È possibile includere nella DLL tutte le schede desiderate o necessarie senza preoccuparsi del fatto che lo stesso nome sia utilizzato in altri wizard. Un altro vantaggio delle DLL consiste nel fatto che è possibile progettare la DLL in modo che non sia dipendente da una sola versione di C++Builder, ma per fare ciò, è assolutamente impossibile utilizzare i package di esecuzione, come viene spiegato nella sezione succesiva.

Utilizzo di una DLL senza package di esecuzione

Non è necessario utilizzare package di esecuzione con una DLL di wizard. Il vantaggio principale nell'utilizzare i package di esecuzione è che la DLL è più piccola. Poiché il wizard è utile solo quando è caricato nell'IDE, si sa quali sono i package di C++Builder disponibili, e pertanto sarà necessario distribuire solo una piccola DLL. D'altro canto, i package sono specifici per ogni versione. Se si desidera distribuire un wizard che funzioni con più versioni di C++Builder e di Delphi, non si devono utilizzare package di esecuzione.

Poiché la variabile *BorlandIDEServices* è definita solo nel package designide, è impossibile utilizzare questa variabile. Invece, è necessario utilizzare il primo parametro passato alla funzione di inizializzazione, che ha lo stesso valore. Il wizard deve salvarlo e utilizzarlo per accedere a tutti i servizi delle API Tools, ad esempio,

```
extern "C" bool __stdcall __declspec (dllexport) INITWIZARD0001(
  const _di_IBorlandIDEServices svc,
  TWizardRegisterProc reg,
  TWizardTerminateProc&)
{
  LocalIDEServices = svc;
  reg(new DocWizard(sUnit));
  reg(new DocWizard(sForm));
  return true;
}

AnsiString __fastcall DocWizard::GetDesigner()
{
  _di_IOTAServices svc;
  LocalIDEServices->Supports(svc);
  return svc->GetActiveDesignerType();
}
```

La modalità di progettazione delle API Tools assicura che la DLL continuerà a funzionare per le nuove versioni di C++Builder. Tutte le vecchie interfacce conservano i rispettivi GUID e le nuove interfacce otterranno i nuovi GUID. Vedere "Numeri di versione dell'interfaccia" a pagina 58-12 per ulteriori informazioni su come le API Tools utilizzano le diverse verioni e i GUID.

Per essere indipendente dalla versione, il wizard non può utilizzare nessuna interfaccia nativa (NTA). Ciò limita quello che il wizard può fare. La limitazione maggiore è che il wizard non può accedere alla barra dei menu e agli altri oggetti dell'IDE. La maggior parte delle interfacce delle API Tools sono interfacce OTA, e pertanto è ancora possibile scrivere wizard utili e interessanti, indipendentemente da come viene installato il wizard.

Appendice



Normativa ANSI per le implementazioni specifiche

Alcuni aspetti del linguaggio ANSI C standard non sono definiti in modo esplicito. Pertanto, ciascun produttore di un compilatore C è libero di definire singolarmente questi aspetti. Questa Appendice descrive come sono stati definiti da Borland questi aspetti. I numeri delle sezioni fanno riferimento al C ANSI/ISO Standard, reso pubblico nel Febbraio 1990.

È bene ricordare che fra il linguaggio C e il linguaggio C++ esistono delle differenze; questa sezione si riferisce solo alla versione C. Per informazioni sulla conformità al linguaggio C++, fare riferimento al sito Web della Borland Community all'indirizzo http://community.borland.com/cpp.

2.1.1.3 Come identificare un diagnostico.

Quando il compilatore viene utilizzato con la combinazione corretta di opzioni, tutti i messaggi inizianti con le parole *Fatal*, *Error* oppure *Warning* sono diagnostici, nel senso indicato dall'ANSI. Le opzioni necessarie per assicurare tale interpretazione sono elencate nella tabella seguente:

Tabella A.1 Opzioni necessarie per assicurare il rispetto delle norme ANSI

Opzione	Effetto
–A	Abilita solo le parole chiave ANSI.
–C–	Abilita solo le parole chiave ANSI.
–i32	Identificatori con almeno 32 caratteri significativi.
–p–	Utilizzo delle convenzioni di chiamata C.
–w–	Disattivazione di tutti i messaggi di segnalazione.
–wbei	Attivazione di messaggi di segnalazione sugli inizializzatori non appropriati.
–wbig	Attivazione di messaggi di segnalazione sulle costanti troppo grandi.
–wcpt	Attivazione di messaggi di segnalazione sul confronto di puntatori non portabili.

Tabella A.1 Opzioni necessarie per assicurare il rispetto delle norme ANSI

Opzione	Effetto
–wdcl	Attivazione di messaggi di segnalazione su dichiarazioni senza tipo o classe di memorizzazione.
–wdup	Attivazione di messaggi di segnalazione su definizioni duplicate di macro non identiche.
–wext	Attivazione di messaggi di segnalazione sulle variabili dichiarate sia external sia static.
–wfdt	Attivazione di messaggi di segnalazione sulla definizione di funzioni mediante un typedef.
–wrpt	Attivazione di messaggi di segnalazione sulla conversione di puntatori non portabili.
–wstu	Attivazione di messaggi di segnalazione su strutture non definite
–wsus	Attivazione di messaggi di segnalazione su conversioni di puntatore sospette.
–wucp	Attivazione di messaggi di segnalazione sull'utilizzo contemporaneo di puntatori a char con e senza segno.
–wvrt	Attivazione di messaggi di segnalazione sulle funzioni void che restituiscono un valore.

Le altre opzioni non indicate esplicitamente possono essere impostate liberamente.

2.1.2.2.1 La semantica degli argomenti per main.

Quando il programma viene eseguito da DOS, *argv*[0] punta al nome del programma.

Le rimanenti stringhe *argv* puntano a ciascun componente degli argomenti della riga comandi DOS. Gli spazi bianchi che separano gli argomenti vengono eliminati e ogni sequenza contigua di caratteri diversi dagli spazi bianchi viene ritenuta un solo argomento. Le stringhe tra apici vengono gestite correttamente (ossia, come una stringa contenente spazi).

2.1.2.3 Che cosa costituisce una periferica interattiva.

Una periferica interattiva è qualsiasi periferica che si presenta come la console.

2.2.1 La sequenza di ordinamento del set di caratteri di esecuzione.

La sequenza di ordinamento del set di caratteri di esecuzione utilizza il valore dei caratteri ASCII.

2.2.1 Elementi dei set di caratteri sorgente e di esecuzione.

I set di caratteri del codice sorgente e il set di caratteri di esecuzione sono costituiti dal set di caratteri esteso ASCII, supportato dai sistemi IBM PC. Ogni carattere diverso da *Ctrl+Z* può apparire in stringhe di testo, nelle costanti di tipo carattere e nei commenti.

2.2.1.2 Caratteri multibyte.

C++Builder sopporta i caratteri multibyte.

2.2.2 Direzione di stampa.

La stampa procede da sinistra a destra, come avviene normalmente nei PC.

2.2.4.2 Numero di bit di un carattere del set di caratteri di esecuzione.

Ogni carattere del set di caratteri di esecuzione è costituito da 8 bit.

3.1.2 Numero di caratteri iniziali significativi degli identificatori.

I primi 250 caratteri sono significativi, ma il numero può venire modificato mediante l'opzione (–i) della riga comandi. Gli identificatori, sia interni che esterni, utilizzano lo stesso numero di caratteri significativi. (Il numero di caratteri significativi degli identificatori di C++ è illimitato.)

3.1.2 Distinzione tra maiuscole e minuscole negli identificatori esterni.

Di norma, il compilatore forza il linker a distinguere i caratteri maiuscoli da quelli minuscoli. Per disattivare tale distinzione, utilizzare l'opzione (–**lc**–) della riga comandi. Nell'IDE è possibile anche utilizzare il comando Project | Options | Advanced Linker e spuntare l'opzione Case-insensitive link.

3.1.2.5 The Rappresentazioni e insiemi di valori dei vari tipi di integer.

Tabella A.2 Identificazione dei diagnostici in C++

Tipo	Valore minimo	Valore massimo
signed char	–128	127
unsigned char	0	255
signed short	–32,768	32,767
unsigned short	0	65,535
signed int	–2,147,483,648	–2,147,483,647
unsigned int	0	4,294,967,295
signed long	–2,147,483,648	2,147,483,647
unsigned long	0	4,294,967,295

Tutti i tipi **char** utilizzano per la memorizzazione un byte di 8 bit.

Tutti i tipi **short** utilizzano 2 byte.

Tutti i tipi **int** utilizzano 4 byte.

Tutti i tipi **long** utilizzano 4 byte.

Se viene richiesto l'allineamento (-**a**), tutti gli oggetti non **char** di tipo integer saranno allineati al byte pari. Se l'allineamento richiesto è -**a4**, il risultato è un allineamento di 4 byte. I tipi carattere non vengono mai allineati.

3.1.2.5 Rappresentazioni e insieme di valori dei tipi di numeri in virgola mobile.

C++Builder adotta, per tutti i tipi di numero in virgola mobile, i formati IEEE in virgola mobile utilizzati dai processori Intel 8086. Il tipo **float** utilizza il formato IEEE

real a 32 bit. Il tipo **double** utilizza il formato IEEE real a 64 bit. Il tipo **long double** utilizza il formato esteso IEEE real a 80 bit.

3.1.3.4 Corrispondenza tra set di caratteri sorgente e di esecuzione.

In un programma in fase di esecuzione, i caratteri contenuti nelle stringhe di testo e nelle costanti carattere rimangono immutati. I set di caratteri sorgente e di esecuzione sono identici.

3.1.3.4 Valore di una costante di testo intera, contenente un carattere o una sequenza di escape non rappresentati nel set di caratteri di esecuzione fondamentale, o nel set di caratteri esteso, per una costante wide character.

I caratteri wide sono supportati.

3.1.3.4 Impostazione locale attiva utilizzata per convertire i caratteri multibyte nei caratteri wide corrispondenti nel caso delle costanti wide character.

Le costanti wide character sono riconosciute.

3.1.3.4 Valore di una costante integer contenente più di un carattere, o di una costante wide character contenente più di un carattere multibyte.

Le costanti di tipo carattere contengono uno o due caratteri. Quando i caratteri sono due, il primo occupa il bit di ordine inferiore della costante, il secondo quello di ordine superiore.

3.2.1.2 Risultato della conversione di un integer in un integer più breve con segno, oppure risultato della conversione di un integer senza segno in un integer di uguale lunghezza con segno, quando il valore non è rappresentabile.

Le conversioni vengono effettuate semplicemente troncando i bit di ordine superiore. I valori con segno vengono memorizzati come valori con complemento a due, per cui il numero risultante viene interpretato come un valore di questo tipo. Quando il bit di ordine superiore dell'integer più piccolo è diverso da zero, il valore viene interpretato come negativo, altrimenti come positivo.

3.2.1.3 Direzione di troncamento quando un numero integrale viene convertito in un numero in virgola mobile che non può rappresentare esattamente il valore originale.

Il valore integer viene arrotondato al più prossimo valore rappresentabile. Quindi, per esempio, il valore **long** (2^{31} –1) viene convertito nel valore **float** 2^{31}. Il rapporto di uguaglianza viene scisso secondo le regole dello standard aritmetico IEEE.

3.2.1.4 Direzione di troncamento o di arrotondamento di un numero in virgola mobile convertito in un numero in virgola mobile più breve.

Il valore integer viene approssimato al più prossimo valore rappresentabile. Il rapporto di uguaglianza viene scisso secondo le regole dello standard aritmetico IEEE.

3.3 Risultato delle operazioni su bit negli integer con segno.

Gli operatori su bit agiscono sugli integer con segno come se fossero i corrispondenti tipi senza segno. Il bit di segno viene trattato come un normale bit di dati. Il risultato viene interpretato come un normale integer con segno in complemento a due.

3.3.2.3 Esito dell'accesso a un membro di un'unione tramite un membro di tipo diverso.

L'accesso è consentito, e il membro di tipo diverso accede ai bit memorizzati. Per capire come accedere a un membro in virgola mobile tramite un membro di tipo diverso, occorre conoscere a fondo come vengono codificati i bit dei valori in virgola mobile. Se il membro memorizzato è più breve del membro utilizzato per accedere al valore, i bit in eccesso mantengono i valori che avevano prima che il membro breve venisse memorizzato.

3.3.3.4 Tipo di integer richiesto per accogliere la dimensione massima di una matrice.

Il tipo delle matrici normali è unsigned int, mentre quello delle matrici di grandi dimensioni è **signed long.**

3.3.4 Esito della conversione di un puntatore in un integer e viceversa.

Nelle conversioni tra integer e puntatori di dimensioni uguali, i bit non subiscono modifiche. Nelle conversioni da un tipo più lungo a uno più corto, vengono troncati i bit di ordine superiore. Nelle conversioni di un tipo integer più breve in un tipo puntatore più lungo, l'integer viene prima esteso a un tipo di integer di dimensione uguale a quella del tipo puntatore.

Così, gli integer con segno estendono il segno in modo da occupare i nuovi byte. Analogamente, i tipi di puntatore di dimensione inferiore, convertiti in tipi integer di dimensione maggiore, vengono prima trasformati in un tipo puntatore di dimensione uguale a quella del tipo integer.

3.3.5 Segno del resto delle divisioni tra integer.

Il resto ha segno negativo quando uno dei due operandi è negativo. Se ambedue gli operandi sono positivi, o negativi, il resto ha segno positivo.

3.3.6 Tipo di integer richiesto per ospitare la differenza tra due puntatori a elementi di una stessa matrice, ptrdiff_t.

Il tipo è **int** con segno.

3.3.7 Risultato di uno spostamento a destra di un tipo integrale con segno negativo.

Quando viene spostato a destra, un valore con segno negativo viene esteso con segno.

3.5.1 Spazio effettivo occupato dagli oggetti nei registri mediante lo specificatore di classe di memorizzazione register.

Gli oggetti dichiarati con tipi integer o puntatore a uno, due o quattro byte, possono essere messi nei registri. Sono disponibili un minimo due e un massimo di sette registri. Il numero di registri effettivamente usati dipende da quali registri sono necessari per ospitare i valori temporanei nella funzione.

3.5.2.1 Interpretazione di un campo di bit semplice come int con segno, oppure come campo di bit int senza segno.

I campi di bit **int** semplici vengono interpretati come campi di bit **signed int**.

3.5.2.1 Ordine di allocazione dei campi di bit all'interno di un int.

I campi di bit vengono allocati dalla posizione di bit di ordine inferiore a quella di ordine superiore.

3.5.2.1 Riempimento (padding) e allineamento dei membri delle strutture.

Per impostazione predefinita, nelle strutture non viene usato il riempimento. Se si attiva l'opzione word alignment (**–a**), le strutture vengono riempite in modo da avere dimensione pari, mentre i membri non di tipo carattere o array di caratteri vengono allineati a un offset multiplo pari.

3.5.2.1 Superamento del limite di una unità di memorizzazione da parte di un campo di bit.

Se non è richiesto l'allineamento (**–a**), i campi di bit possono superare i limiti dword, ma non vengono mai memorizzati in più di quattro byte adiacenti.

3.5.2.2 Scelta del tipo di integer per rappresentare i valori di un tipo enumerazione.

Tutti gli enumeratori vanno memorizzati come **int** completi. Se i valori non sono contenuti in un **int**, si memorizzano in un tipo **long** oppure **unsigned long**. Questo è il comportamento di default, come specificato dall'opzione **–b** del compilatore.

Il comportamento **–b**- specifica che le enumerazioni vanno preferibilmente memorizzate nel tipo integer più piccolo in grado di rappresentare i valori. Questo include tutti i tipi integrali come, per esempio, **signed char**, **unsigned char**, **signed short**, **unsigned short**, **signed int**, **unsigned int**, **signed long** e **unsigned long**.

Per compatibilità con C++, va specificata l'opzione **–b**-. Infatti, in C++ non è corretto memorizzare tutte le enumerazioni come **int**.

3.5.3 Accesso a un oggetto di tipo volatile-qualified.

Ogni riferimento a un oggetto volatile ha accesso all'oggetto. La possibilità che l'accesso a locazioni di memoria adiacenti consenta anche l'accesso a un oggetto dipende dall'impostazione fisica della memoria a livello hardware. Nel caso di dispositivi di memoria particolari (per esempio, la memoria video) dipende da come il dispositivo è stato progettato. Per la normale memoria del PC, gli oggetti volatili vengono utilizzati solo per la memoria cui si può accedere mediante interrupt asincroni, per cui l'accesso a oggetti adiacenti non ha effetto.

3.5.4 Massimo numero di dichiaratori che possono modificare un tipo aritmetico, di struttura o di unione.

Non esiste un limite specifico al numero di dichiaratori. Il numero ammissibile di dichiaratori è piuttosto grande, ma si riduce quando i dichiaratori sono annidati profondamente in un insieme di blocchi di una funzione. Il numero consentito a livello di file è almeno 50.

3.6.4.2 Massimo numero di valori case in un'istruzione switch.

Non esiste un limite specifico al numero di istruzioni case in uno switch. Finché c'è memoria disponibile per contenere le istruzioni case, il compilatore le accetta.

3.8.1 Corrispondenza tra il valore di una costante di tipo carattere di un solo carattere in un'espressione costante che controlla le inclusioni condizionali e il valore della stessa costante nel set di caratteri di esecuzione. Possibilità o meno della costante di avere valore negativo.

Tutte le costanti di tipo carattere, anche quelle presenti nelle direttive condizionali, utilizzano il medesimo set di caratteri (esecuzione). Le costanti carattere di un solo carattere sono negative se il tipo carattere è con segno (default e **–K** non richiesti).

3.8.2 Metodo per localizzare i file sorgente includibili.

Per i nomi di file include racchiusi tra parentesi ad angolo, se le directory include vengono immesse dalla riga comandi, i file vengono ricercati in tali directory. La ricerca nelle directory avviene nel seguente ordine:

1. Directory specificate nella riga comandi.
2. Directory specificate in BCC32.CFG.
3. Directory attuale se non viene specificata alcuna directory include.

3.8.2 Supporto dei nomi racchiusi da doppi apici dei file sorgente incorporabili.

La ricerca dei nomi di file incorporabili racchiusi da doppi apici avviene nel seguente ordine:

1. La stessa directory del file che contiene l'istruzione **#include**.
2. Le directory dei file che includono (**#include**) quel file.
3. La directory attuale.
4. Il percorso specificato dall'opzione /**I** del compilatore.
5. I percorsi specificati dalla variabile d'ambiente INCLUDE.

3.8.2 Relazione tra sequenze di caratteri e nomi dei file sorgente.

Le barre rovesciate nei nomi dei file include sono interpretate come caratteri distinti, non come caratteri di escape. Non viene fatta distinzione tra lettere maiuscole e minuscole.

3.8.8 Definizione di __DATE__ e di __TIME__ quando non sono disponibili.

Data e ora sono sempre disponibili e utilizzano la data e l'ora di sistema.

4.1.1 Carattere di separazione decimale.

Il carattere di separazione decimale è il punto (.).

4.1.5 Tipo dell'operatore sizeof, *size_t*.

Il tipo di *size_t è unsigned*.

4.1.5 Costante puntatore nullo alla quale si espande la macro NULL.

NULL si espande in uno 0 int, o in uno 0 long. Entrambi sono numeri a 32 bit signed.

4.2 Messaggi diagnostici visualizzati e comportamento di terminazione della funzione assert.

Il messaggio diagnostico visualizzato è "Assertion failed: *espressione*, file *nomefile*, line *nn*", dove *espressione* è l'espressione che non ha superato la valutazione, *nomefile* è il nome del file sorgente e *nn* è il numero della riga dove la valutazione è avvenuta.

Immediatamente dopo il messaggio diagnostico viene chiamato **Abort**.

4.3 Aspetti implementativi delle funzioni di verifica dei caratteri e di corrispondenza maiuscole/minuscole.

Vedere la sezione 4.3.1.

4.3.1 Set di caratteri testati dalle funzioni isalnum, isalpha, iscntrl, islower, isprint e isupper.

I primi 128 caratteri ASCII dell'impostazione locale C predefinita. In alternativa, tutti i 256 caratteri.

4.5.1 Valori restituiti dalle funzioni matematiche in caso di errori di dominio.

Viene restituito uno IEEE NAN (non un numero).

4.5.1 Le funzioni matematiche impostano l'espressione integer *errno* al valore della macro ERANGE negli errori di underflow?.

No, solo per altri errori - dominio, singolarità, overflow, perdita totale della precisione.

4.5.6.4 Quando la funzione fmod ha come secondo argomento zero, si verifica un errore di domino o viene restituito zero?

No; fmod(x,0) restituisce 0.

4.7.1.1 Insieme dei segnali della funzione signal.

SIGABRT, SIGFPE, SIGILL, SIGINT, SIGSEGV, e SIGTERM.

4.7.1.1 Semantica di ciascun segnale riconosciuto dalla funzione signal.

Vedere la descrizione di signal.

4.7.1.1 Gestione di default e gestione in fase di avvio del programma dei segnali riconosciuti dalla funzione signal.

Vedere la descrizione di signal.

4.7.1.1 Se l'equivalente di signal(sig, SIG_DFL); non viene eseguito prima della chiamata di un gestore di segnale, quale blocco del segnale viene messo in atto.

L'equivalente di signal(sig, SIG_DFL) viene eseguito sempre.

4.7.1.1 In caso di ricevimento del segnale SIGILL da parte del gestore specificato dalla funzione signal, la gestione di default viene resettata oppure no.

No, non viene resettata.

4.9.2 Obbligatorietà di far seguire un carattere di nuova riga all'ultima riga di un flusso di testo.

Il carattere di nuova riga non è necessario.

4.9.2 I caratteri spazio, scritti in un flusso di testo, che precedono immediatamente il carattere di nuova riga, compaiono o no quando vengono letti.

I caratteri compaiono.

4.9.2 Numero di caratteri null aggiungibili ai dati scritti in stream binari.

Nessuno

4.9.3 Posizionamento iniziale all'inizio o alla fine di un file dell'indicatore di posizione del file di un flusso in modalità append.

Inizialmente, l'indicatore di posizione del file di uno stream in modalità append viene situato all'inizio del file. Viene portato alla fine del file prima di ogni operazione di scrittura.

4.9.3 Troncamento del file associato a un flusso di testo dopo il punto in cui avviene una scrittura.

Una scrittura di 0 byte tronca o non tronca il file a seconda dell'impostazione del buffer del file. Conviene stabilire che il comportamento di una scrittura di 0 byte è indeterminato.

4.9.3 Caratteristiche del buffer dei file.

I file possono avere un buffer completo, un buffer di riga, oppure essere privi di buffer. Se il file ha il buffer, all'apertura viene creato un buffer predefinito di 512 byte.

4.9.3 Esistenza effettiva dei file di lunghezza zero.

I file di lunghezza zero esistono.

4.9.3 Possibilità di aprire più volte uno stesso file.

È consentito aprire più volte un file.

4.9.4.1 Effetto della funzione remove su un file aperto.

Non viene controllato specificamente se un file è già aperto; la responsabilità del controllo spetta al programmatore.

4.9.4.2 Controllo di esistenza del nome, prima di una chiamata di rename.

Rename restituisce –1 ed *errno* viene impostato a EEXIST.

4.9.6.1 Output delle conversioni %p in fprintf.

L'output è composto da otto cifre esadecimali (XXXXXXXX), riempite con zero, in lettere maiuscole (come %08lX).

4.9.6.2 Input delle conversioni %p in fscanf.

Vedere 4.9.6.1.

4.9.6.2 Interpretazione del carattere – (hyphen) quando non è né il primo né l'ultimo carattere dell'elenco di scansione, nelle conversioni %[in fscanf.

Vedere la descrizione di scanf.

4.9.9.1 Valore della macro errno, impostato in caso di insuccesso dalle funzioni fgetpos o ftell.

EBADF Numero di file errato.

4.9.10.4 Messaggi generati da perror.

Messaggi generati in Win32	
Arg list too big	Math argument
Attempted to remove current directory	Memory arena trashed
Bad address	Name too long
Bad file number	No child processes
Block device required	No more files
Broken pipe	No space left on device
Cross-device link	No such device
Error 0	No such device or address
Exec format error	No such file or directory
Executable file in use	No such process
File already exists	Not a directory
File too large	Not enough memory
Illegal seek	Not same device
Inappropriate I/O control operation	Operation not permitted
Input/output error	Path not found
Interrupted function call	Permission denied
Invalid access code	Possible deadlock
Invalid argument	Read-only file system
Invalid data	Resource busy
Invalid environment	Resource temporarily unavailable
Invalid format	Result too large
Invalid function number	Too many links
Invalid memory block address	Too many open files
Is a directory	

4.10.3 Comportamento di calloc, malloc, o realloc quando la dimensione richiesta è zero.

calloc e malloc ignorano la richiesta e restituiscono 0; realloc libera il blocco.

4.10.4.1 Comportamento della funzione abort nei confronti dei file aperti e dei file temporanei.

I buffer dei file non vengono scaricati e i file non vengono chiusi.

4.10.4.3 Stato restituito da exit (EXIT_SUCCESS, oppure EXIT_FAILURE), se il valore dell'argomento è diverso da zero.

Nessuna segnalazione particolare. Lo stato viene rappresentato da un **signed char.**

4.10.4.4 Set dei nomi ambiente e metodo utilizzato per alterare l'elenco di ambiente utilizzato da getenv.

Le stringhe di ambiente sono quelle definite nel sistema operativo mediante il comando SET. Volendo modificare le stringhe per la durata del programma attivo, si

ricorre a *putenv*, mentre per modificare permanentemente le stringhe di ambiente bisogna utilizzare il comando SET.

4.10.4.5 Il contenuto e i modi di esecuzione delle stringhe mediante le funzioni di sistema.

La stringa viene interpretata come un comando del sistema operativo. Vengono utilizzati COMSPEC o CMD.EXE e l'argomento stringa viene passato come un comando da eseguire. Oltre a tutti i comandi del sistema operativo, possono essere eseguiti i file batch e i programmi eseguibili.

4.11.6.2 Contenuto delle stringhe dei messaggi di errore restituite da strerror.

Vedere 4.9.10.4.

4.12.1 Fuso orario locale e Daylight Saving Time (ora legale).

Definiti in base alla data e all'ora locali del PC.

4.12.2.1 Base dei tempi.

È misurata da tick del clock; l'origine è l'istante in cui inizia l'esecuzione del programma.

4.12.3.5 Formati della data e dell'ora.

C++Builder applica i formati ANSI.

Appendice

B

Guida di riferimento agli script lato server di WebSnap

Questa appendice spiega il meccanismo con cui lo script viene utilizzato dagli adapter WebSnap per generare pagine HTML dinamiche in applicazioni per Web server WebSnap. L'appendice è pensata per gli sviluppatori che nei moduli di pagine web utilizzano produttori di pagine invece di adapter produttori di pagine. Gli adapter produttori di pagine gestiscono la generazione di script automaticamente; i normali produttori di pagine necessitano di script aggiunti manualmente nei loro modelli di pagina. Le informazioni fornite in questa appendice aiuteranno a scrivere lo script nei modelli di pagina.

Questa appendice potrebbe interessare anche agli utenti di adapter produttori di pagine che vogliono capirne meglio l'output. Tuttavia, la manipolazione di script è un argomento complesso. Non è necessario capire come generare script per scrivere applicazioni di base WebSnap.

Questa appendice è divisa in tre sezioni. La prima sezione spiega i vari tipi di oggetto a cui è possibile accedere nello script. La seconda sezione tratta oggetti globali definiti nelle applicazioni WebSnap. La terza sezione contiene esempi JScript che mostrano come si può utilizzare lo script nei modelli delle pagine HTML per estrarre informazioni dall'applicazione per Web server.

Questa appendice vuole essere un riferimento API per le interfacce di script degli oggetti. Le descrizioni delle proprietà dell'oggetto includono il nome della proprietà, il tipo di proprietà (come text o Boolean) e specificano se la proprietà può essere letta o scritta. Le descrizioni del metodo iniziano con il nome del metodo e con la sintassi chiamante.

Tipi di oggetto

La Tabella B.1 elenca i tipi di oggetto generali in grado di gestire script. Questi tipi vengono comunemente trovati come proprietà di oggetti come gli oggetti globali. Non sono elencati tutti i tipi di oggetto in grado di gestire script, ma solo quei tipi le cui istanze possono avere nomi differenti. Ad esempio, esiste un tipo di oggetto application. Poiché esiste un solo tipo di oggetto application istanziato in ogni applicazione, il tipo application è descritto come oggetto Application nella sezione realtiva aglioggetti globali.

Tabella B.1 Tipi di oggetto di WebSnap

Tipo di oggetto	Descrizione
Tipo Adapter (page B-2)	Definisce le proprietà e i metodi di un adapter.È possibile accedere agli adapter tramite nome come proprietà di un Module.
TipoAdapterAction (page B-4)	Definisce le proprietà e i metodi di un'azione di un adapter. Le azioni sono indicate da un nome come proprietà di un adapter.
Tipo AdapterErrors (page B-6)	Definisce la proprietà Errors di un adapter. La proprietà Errors è utilizzata per elencare errori che si sono verificati durante l'esecuzione di un'azione o la generazione di una pagina.
Tipo AdapterField (page B-7)	Definisce le proprietà e i metodi di un campo di un adapter. I campi sono indicati da un nome come proprietà di un Adapter.
Tipo AdapterFieldValues (page B-11)	Definisce le proprietà e i metodi della proprietà Values di un campo di un adapter.
Tipo AdapterFieldValuesList (page B-11)	Definisce la proprietà e i metodi della proprietà ValuesList di un campo di un adapter.
Tipo AdapterHiddenFields (page B-12)	Definisce la proprietà HiddenFields e HiddenRecordFields di un adapter.
Tipo AdapterImage (page B-12)	Definisce la proprietà Image dei campi di un adapter e delle azioni di un adapter.
Tipo Module (page B-12)	Definisce le proprietà di un Module. È possibile accedere a un Module tramite nome come proprietà della variabile Modules.
Tipo Page (page B-13)	Definisce le proprietà di una pagina. È possibile accedere a una pagina tramite nome come proprietà dell'oggetto Pages. È possibile accedere alla pagina che viene generata utilizzando l'oggetto Page.

Tipo Adapter

Definisce le proprietà e i metodi di un adapter. È possibile accedere ad adapter tramite nome come proprietà di un modulo (per esempio ModuleName.Adapter).

Gli adapter contengono i componenti campo e i componenti azione che rappresentano rispettivamente i dati e i comandi. Le istruzioni di uno script lato server accedono al valore dei campi dell' adapter e ai parametri delle azioni dell'adapter per costruire le schede e le tabelle HTML.

Proprietà

Actions: Enumerator

Vedere anche: proprietà *Fields* del tipo Adapter (di seguito), Esempio 8 (page B-23)

Elenca gli oggetti azione. Utilizzare la proprietà *Actions* per esaminare in sequenza tutte le azioni di un adapter.

CanModify: Booleano, lettura

Vedere anche: proprietà *CanView* del tipo Adapter (sotto) e del tipo AdapterField (page B-7)

Indica se l'utente possiede l'autorizzazione per modificare i campi di questo adapter. Utilizzare la proprietà CanModify per generare dinamicamente HTML che è sensibile ai diritti dell'utente. Ad esempio, una pagina può includere un elemento <input> se *CanModify* è *True* oppure <p> se *CanModify* è *False*.

CanView: Booleano, lettura

Vedere anche: *CanModify* del tipo Adapter (sopra) e del tipo AdapterField (page B-7)

Indica se l'utente ha l'autorizzazione per esaminare i campi di questo adapter. Utilizzare la proprietà *CanModify* per generare dinamicamente HTML che è sensibile ai diritti dell'utente.

ClassName_: testo, a sola lettura

Vedere anche: *Name_* (di seguito)

Identifica il nome della classe del componente adapter.

Errors: AdapterErrors, lettura

Vedere anche: Tipo AdapterErrors (page B-6), Esempio 7 (page B-23)

Elenca gli errori rilevati durante il trattamento di una richiesta HTTP. Gli adapter catturano gli errori che si verificano durante la generazione di una pagina HTML o l'esecuzione di un'azione dell'adapter. Utilizzare la proprietà *Errors* per elencare gli errori e visualizzare i messaggi di errore su una pagina HTML.

Fields: Enumerator

Vedere anche: *Actions*

Elenca gli oggetti campo. Utilizzare la proprietà Fields per esaminare in sequenza i campi di un adapter.

HiddenFields: AdapterHiddenFields

Vedere anche: *HiddenRecordFields*, tipo AdapterHiddenFields (page B-12), Esempio 10 (page B-25), Esempio 22 (page B-37)

Definisce i campi di input non visualizzati che passano informazioni sullo stato dell'adapter. Un esempio di informazioni di stato è una modalità di *TDataSetAdapter*. Edit e Insert sono due possibili valori della modalità. Quando *TDataSetAdapter* è utilizzato per generare una scheda HTML, la proprietà *HiddenFields* definisce un campo non visualizzato per quella modalità. Quando la

scheda HTML viene inviata, la richiesta HTTP contiene questo valore del campo nascosto. Quando viene eseguita un'azione, il valore della modalità viene estratto dalla richiesta HTTP. Se la modalitàè Insert, nel dataset viene inserita una nuova riga. Se la modalità è Edit, viene aggiornata una riga di dataset.

HiddenRecordFields: AdapterHiddenFields

Vedere anche: *HiddenFields*, tipo AdapterHiddenFields (page B-12), Esempio 10 (page B-25), Esempio 22 (page B-37)

Definisce i campi di input non visualizzati che passano le informazioni di stato necessarie per ogni riga o record nella scheda HTML. Ad esempio, quando *TDataSetAdapter* è utilizzato per generare una scheda HTML, la proprietà *HiddenRecordFields* definirà un campo non visualizzato che identifica un valore chiave per ogni riga in una tabella HTML. Quando la scheda HTML viene inoltrata, la richiesta HTTP conterrà questi valori del campo nascosto. Quando viene eseguita un'azione che aggiorna più righe in un dataset, *TDataSetAdapter* utilizza questi valori chiave per individuare le righe da aggiornare.

Mode: testo, lettura/scrittura

Vedere anche: Esempio 10 (page B-25)

Imposta oppure ottiene la modalità dell'adapter.

Alcuni adapter supportano una modalità. Ad esempio, *TDataSetAdapter* supporta le modalità Edit, Insert, Browse e Query. La modalità influisce sul comportamento dell'adapter. Quando *TDataSetAdapter* è in modalità Edit, l'inoltro di una scheda aggiorna una riga in una tabella. Quando *TDataSetAdapter* è in modalità Insert, l'inoltro di una scheda inserisce una riga in una tabella.

Name_: testo, lettura

Identifica il nome variabile dell'adapter.

Records: Enumerator, lettura

Vedere anche: Esempio 9 (page B-24)

Elenca i record dell'adapter. Utilizzare la proprietà Records per esaminare in sequenza i record dell'adapter per generare una tabella HTML.

Tipo AdapterAction

Vedere anche: Tipo Adapter (page B-2),tipo AdapterField (page B-7)

Il tipo AdapterAction definisce le proprietà e i metodi di un'azione dell'adapter.

Proprietà

Array: Enumerator

Vedere anche: Esempio 11 (page B-26)

Elenca i comandi di un'azione dell'adapter. Utilizzare la proprietà *Array* per esaminare in sequenza i comandi. *Array* è *Null* se l'azione non supporta comandi multipli.

TAdapterGotoPageAction è un esempio di un'azione con comandi multipli. Questa azione ha un comando per ogni pagina definita dall'adapter genitore. La proprietà *Array* è utilizzata per generare una serie di collegamenti ipertestuali in modo che l'utente possa fare clic su uno di essi per spostarsi a un'altra pagina.

AsFieldValue: testo, lettura

Vedere anche: *AsHREF*, Esempio 10 (page B-25), Esempio 21 (page B-36)

Fornisce un valore di testo che può essere inviato in un campo non visualizzato.

AsFieldValue identifica il nome e i parametri dell'azione. Immettere questo valore in un campo nascosto di nome __act. Quando la scheda HTML viene inoltrata, il dispatcher dell'adapter può estrarre il valore dalla richiesta HTTP e utilizzarlo per individuare e chiamare l'azione dell'adapter.

AsHREF: testo, lettura

Vedere anche: *AsFieldValue*, Esempio 11 (page B-26)

Fornisce un valore del testo che può essere utilizzato come valore dell'attributo href in un marcatore <a>.

AsHREF identifica il nome e i parametri dell'azione. Immettere questo valore in un marcatore anchor per richiedere di eseguire questa azione. Si noti che un marcatore anchor su una scheda HTML non permetterà l'inoltro della scheda. Se l'azione utilizza valori inviati della scheda, utilizzare un campo di scheda nascosto e *AsFieldValue* per identificare l'azione.

CanExecute: Booleano, lettura

Indica se l'utente ha i diritti necessari per eseguire questa azione.

DisplayLabel: testo, lettura

Vedere anche: Esempio 21 (page B-36)

Suggerisce un'etichetta HTML di visualizzazione per questa azione dell'adapter.

DisplayStyle: stringa, lettura

Vedere anche: Esempio 21 (page B-36)

Suggerisce uno stile HTML di visualizzazione per questa azione.

Lo script lato server può utilizzare *DisplayStyle* per determinare come generare il codice HTML. Gli adapter nativi possono restituire uno dei seguenti stili di visualizzazione:

Valore	Significato
''	Stile di visualizzazione non definito
'Button'	Viene visualizzato come <input type="submit">
'Anchor'	Utilizza <a>

Enabled: Booleano, lettura

Vedere anche: Esempio 21 (page B-36)

Indica se questa azione dovrebbe essere abilitata sulla pagina HTML.

Name: stringa, lettura

Fornisce il nome della variabile di questa azione dell'adapter

Visible: Booleano, lettura

Indica se questo campo dell'adapter dovrebbe essere visibile sulla pagina HTML.

Metodi

LinkToPage(PageSuccess, PageFail): AdapterAction, lettura

Vedere anche: Esempio 10 (page B-25), Esempio 11 (page B-26), Esempio 21 (page B-36), oggetto Page , tipo AdapterAction (page B-4)

Utilizzare *LinkToPage* per specificare le pagine che devono essere visualizzate dopo che l'azione è stata eseguita. Il primo parametro è il nome della pagina da visualizzare se l'azione è stata eseguita con successo. Il secondo parametro è il nome della pagina da visualizzare se si verificano errori durante l'esecuzione

Tipo AdapterErrors

Vedere anche: proprietà *Errors* del tipo Adapter (page B-2)

Il tipo *AdapterErrors* definisce le proprietà della proprietà *Errors* di un adapter.

Proprietà

Field: AdapterField, lettura

Vedere anche: Tipo AdapterField (page B-7)

Identifica il campo dell'adapter che ha causato un errore.

Questa proprietà è *Null* se l'errore non è associato a un particolare campo dell'adapter.

ID: intero, lettura

Fornisce l'identificativo numerico per un errore.

Questa proprietà vale zero se non è stato definito un ID.

Message: testo, lettura

Vedere anche: Esempio 7 (page B-23)

Fornisce una descrizione testuale dell'errore.

Tipo AdapterField type

Vedere anche: Tipo Adapter (page B-2), Tipo AdapterAction (page B-4)

Il tipo AdapterField definisce le proprietà e i metodi di un campo dell'adapter.

Proprietà

CanModify: Booleano, lettura

Vedere anche: proprietà *CanView* del tipo AdapterField (di seguito) e del tipo Adapter (page B-2)

Indica se l'utente ha i diritti necessari per modificare il valore di questo campo.

CanView: Booleano, lettura

Vedere anche: proprietà *CanModify* del tipo AdapterField (di seguito) e del tipo Adapter (page B-2)

Indica se l'utente ha i diritti necessari per visualizzare il valore di questo campo.

DisplayLabel: testo, lettura

Suggerisce un'etichetta di visualizzazione HTML per questo campo dell'adapter.

DisplayStyle: testo, lettura

Vedere anche: *InputStyle* (di seguito), *ViewMode* (di seguito), Esempio 17 (page B-34)

DisplayStyle suggerisce come visualizzare una rappresentazione a sola lettura del valore di un campo.

Lo script lato server può utilizzare *DisplayStyle* per determinare come generare HTML. Un campo dell'adapter può restituire uno dei seguenti stili di visualizzazione:

Valore	Stile di visualizzazione HTML
''	Indefinito.
'Text'	Utilizza <p>.
'Image'	Utilizza <img>. La proprietà Image del campo definisce la proprietà src.
'List'	Utilizza <UL>. Elenca la proprietà Values per generare ogni elemento <LI>.

La proprietà *ViewMode* indica se utilizzare *InputStyle* o *DisplayStyle* per generare il codice HTML.

DisplayText: testo, lettura

Vedere anche: *EditText* (di seguito), Esempio 9 (page B-24)

Fornisce il testo da utilizzare quando viene visualizzato il valore del campo dell'adapter per la sola lettura. Il valore di DisplayText può includere una formattazione numerica.

DisplayWidth: intero, lettura

Vedere anche: *MaxLength* (di seguito)

Suggerisce una larghezza di visualizzazione, in caratteri, per il valore di un campo dell'adapter.

Se la larghezza di visualizzazione non è definita viene restituito -1.

EditText: testo, lettura

Vedere anche: *DisplayText* (sopra), Esempio 10 (page B-25)

Fornisce il testo da utilizzare quando viene definito un input HTML per questo campo dell'adapter. Di solito il valore di EditText non è formattato.

Image: Tipo AdapterImage, lettura

Vedere anche: Tipo AdapterImage (page B-12), Esempio 12 (page B-28)

Fornisce un oggetto che definisce un'immagine per questo campo dell'adapter.

Viene restituito Null se il campo dell'adapter non fornisce un'immagine.

InputStyle: testo, lettura

Vedere anche: *DisplayStyle* (sopra), *ViewMode* (sotto), Esempio 17 (page B-34)

Suggerisce uno stile di input HTML per questo campo.

Lo script lato server può utilizzare InputStyle per determinare come generare un elemento HTML. Un campo dell'adapter può restituire uno dei seguenti stili di input:

Valore	Significato
''	Stile di input indefinito.
'TextInput'	Utilizza <input type="text">.
'PasswordInput'	Utilizza <input type="password">.
'Select'	Utilizza <select>. Elenca la proprietà ValuesList per generare ogni elemento <option>.
'SelectMultiple'	Utilizza <select multiple>. Elenca la proprietà ValuesList per generare ogni elemento <option>.
'Radio'	Enumera la proprietà ValuesList per generare uno o più <input type="radio">.
'CheckBox'	Enumera la proprietà ValuesList per generare uno o più <input type="checkbox">.
'TextArea'	Utilizza <textarea>.
'File'	Utilizza <input type="file">.

La proprietà *ViewMode* indica se utilizzare *InputStyle* o *DisplayStyle* per generare il codice HTML.

InputName: testo, lettura

Vedere anche: Esempio 10 (page B-25)

Fornisce un nome a un elemento di input HTML per modificare questo campo dell'adapter.

Utilizzare *InputName* quando si genera un elemento HTML <input>, <select> o <textarea> per permettere al componente adapter di associare le coppie nome/valore nella richiesta HTTP con i campi dell'adapter.

MaxLength: intero, lettura

Vedere anche: *DisplayWidth* (sopra)

Indica la lunghezza massima in caratteri che può essere immessa in questo campo.

MaxLength è -1 se la lunghezza massima non è definita.

Name: testo, lettura

Restituisce il nome della variabile del campo dell'adapter.

Required: Booleano, lettura

Indica se viene richiesto un valore per questo campo dell'adapter quando viene inoltrata una scheda.

Value: variant, lettura

Vedere anche: *Values* (di seguito), *DisplayText* (sopra), *EditText* (sopra)

Restituisce un valore che può essere utilizzato nei calcoli. Ad esempio, utilizzare *Value* quando si sommano due valori di campo dell'adapter.

Values: AdapterFieldValues, lettura

Vedere anche: *ValuesList* (di seguito), tipo AdapterFieldValues (page B-10), *Value* (sopra), Esempio 13(page B-28)

Restituisce un elenco dei valori del campo. La proprietà *Values* è Null a meno che questo campo dell'adapter non supporti più valori. Un campo con valori multipli verrebbe utilizzato, ad esempio, per permettere all'utente di scegliere più valori in un elenco.

ValuesList: AdapterFieldValuesList, lettura

Vedere anche: *Values* (sopra), tipo AdapterFieldValuesList (page B-11), Esempio 13 (page B-28)

Fornisce un elenco di scelte per questo campo dell'adapter. Utilizzare *ValuesList* quando si genera una lista di selezione, un gruppo di caselle di controllo o un gruppo di pulsanti di scelta HTML. Ogni elemento in *ValuesList* ha un valore e può avere un nome.

Visible: Booleano, lettura

Indica se questo campo dell'adapter dovrebbe essere visibile sulla pagina HTML.

ViewMode: testo, lettura

Vedere anche: *DisplayStyle* (sopra), *InputStyle* (sopra), Esempio17 (page B-34)

Suggerisce come visualizzare questo valore di campo dell'adapter su una pagina HTML.

Un campo dell'adapter può restituire uno dei seguenti modi di visualizzazione:

Valore	Modo di visualizzazione
''	Indefinito.
'Input'	Genera elementi di scheda HTML modificabili utilizzando <input>, <textarea> o <select>.
'Display'	Genera HTML a sola lettura utilizzando <p>, o <img>.

La proprietà *ViewMode* indica se utilizzare *InputStyle* o *DisplayStyle* per generare il codice HTML.

Metodo

IsEqual(Valore): Booleano

Vedere anche: Esempio 16 (page B-32)

Chiamare questa funzione per confrontare una variabile con il valore di un campo dell'adapter.

Tipo AdapterFieldValues

Vedere anche: proprietà *Values* del tipo AdapterField (page B-7)

Fornisce un elenco dei valori del campo. Questa proprietà è supportata dai campi a valori multipli dell'adapter. Un campo a valori multipli verrebbe utilizzato, ad esempio, per permettere all'utente di selezionare più valori in una lista di selezione.

Proprietà

Records: Enumerator, lettura

Vedere anche: Esempio 15 (page B-31)

Elenca i record nell'elenco di valori.

Value: variant, lettura

Vedere anche: *ValueField* (di seguito)

Restituisce il valore dell'elemento di enumerazione corrente.

ValueField: AdapterField, lettura

Vedere anche: Tipo AdapterField (page B-7), Esempio 15 (page B-31)

Restituisce un campo dell'adapter per l'elemento di enumerazione corrente. Utilizzare *ValueField*, ad esempio, per ottenere *DisplayText* dell'elemento di enumerazione corrente.

Metodi

HasValue(Value): Booleano

Vedere anche: Esempio 14 (page B-29)

Indica se un certo valore è nell'elenco di valori del campo. Questo metodo è utilizzato per determinare se scegliere un elemento in una lista di selezione HTML oppure se selezionare un elemento in un gruppo di caselle di controllo.

Tipo AdapterFieldValuesList

Vedere anche: Tipo Adapter (page B-2)

Fornisce un elenco dei possibili valori per questo campo dell'adapter.

Utilizzare *ValuesList* quando si genera una lista di selezione, un gruppo di caselle di controllo o un gruppi di pulsanti di opzione HTML. Ogni elemento in *ValuesList* contiene un valore e può contenere un nome.

Proprietà

Image: AdapterImage, lettura

Restituisce l'immagine dell'elemento di enumerazione corrente, oppure Null se l'elemento non ha un'immagine.

Records: Enumerator, lettura

Elenca i record nell'elenco di valori.

Value: variant, lettura

Restituisce il valore dell'elemento di enumerazione corrente.

ValueField: AdapterField, lettura

Vedere anche: Tipo AdapterField (page B-7), Esempio 15 (page B-31)

Restituisce un campo dell'adapter per l'elemento di enumerazione corrente. Utilizzare *ValueField*, ad esempio, per ottenere *DisplayText* per l'elemento di enumerazione corrente.

ValueName: testo, lettura

Restituisce il nome di testo dell'elemento corrente. *ValueName* è vuoto se il valore non ha un nome.

Metodi

ImageOfValue(Value): AdapterImage

Ricerca l'immagine associata a questo valore. Restituisce Null se non vi sono immagini.

NameOfValue(Value): testo

Ricerca il nome asociato a questo valore. Restituisce una stringa vuota se il valore non viene trovato o se il valore non ha un nome.

Tipo AdapterHiddenFields

Vedere anche: proprietà *HiddenFields* e *HiddenRecordFields* del tipo Adapter (page B-2)

Fornisce accesso ai nomi e ai valori di campo nascosti che un adapter richiede su schede HTML utilizzate per inviare modifiche.

Proprietà

Name: testo, lettura

Restituisce il nome del campo nascosto che viene elencato.

Value: testo, lettura

Restituisce il valore della stringa del campo nascosto che viene elencato.

Metodi

WriteFields(Response)

Vedere anche: Esempio 10 (page B-25), Esempio 22 (page B-37)

Scrive i nomi e i valori del campo nascosto utilizzando <input type=”hidden”>.

Chiamare questo metodo per scrivere tutti i campi nascosti HTML su una scheda HTML.

Tipo AdapterImage

Vedere anche: Tipo AdapterField (page B-12),tipo AdapterAction (page B-4)

Rappresenta un’immagine associata a un’azione o a un campo.

Proprietà

AsHREF: testo, lettura

Vedere anche: Esempio 11 (page B-26), Esempio 12 (page B-28)

Fornisce un’URL che può essere utilizzata per definire un elemento HTML <img>.

Tipo di modulo

Vedere anche: oggetto Modules (page B-16)

I componenti adapter possono essere contraddistinti da un nome come proprietà di un modulo. Utilizzare un modulo anche per elencare gli oggetti in grado di gestire script (di solito adapter) di un modulo.

Proprietà

Name_: testo, lettura

Vedere anche: Esempio 20 (page B-36)

Identifica il nome della variabile del modulo. Questo è il nome utilizzato per accedere al modulo come proprietà della variabile Modules.

ClassName_: testo, lettura

Vedere anche: Esempio 20 (page B-36)

Identifica il nome della classe del modulo.

Objects: Enumerator

Vedere anche: Esempio 20 (page B-36)

Utilizzare *Objects* per elencare gli oggetti in grado di gestire script (di solito adapter) all'interno di un modulo.

Tipo Page

Vedere anche: oggetto Page (page B-17), Esempio 20 (page B-36)

Definisce le proprietà e i metodi di pagine.

Proprietà

CanView: Booleano, lettura

Indica se l'utente ha i diritti necessari per esaminare questa pagina.

Una pagina registra i diritti di accesso. *CanView* confronta i diritti registrati dalla pagina con i diritti concessi all'utente.

DefaultAction: tipo AdapterAction , lettura

Vedere anche: Esempio 6 (page B-22)

Identifica l'azione predefinita dell'adapter azione associata a questa pagina.

Un'azione predefinita è di solito utilizzata quando i parametri devono essere passati a una pagina. *DefaultAction*può essere Null.

HREF: testo, lettura

Vedere anche:: Esempio 5 (page B-22)

Fornisce un'URL che può essere utilizzata per generare un collegamento ipertestuale con questa pagina utilizzando il marcatore <a>.

LoginRequired: Booleano, lettura

Indica se l'utente deve eseguireil login prima di accedere a questa pagina.

Una pagina registra un flag *LoginRequired*. Se è *True* all'utente non è consentito accedere a questa pagina a meno che non effettui il login.

Name: testo, lettura

Vedere anche: Esempio 5 (page B-22)

Fornisce il nome della pagina registrata.

Se la pagina è pubblicata, *PageDispatcher* genera la pagina nel caso in cui il nome della pagina è un suffisso dell'informazione di percorso della richiesta HTTP.

Published: Booleano, lettura

Vedere anche: Esempio 5 (page B-22)

Indica se l'utente può accedere a questa pagina specificando il nome della pagina come suffisso dell'URL.

Una pagina registra un flag *Published*. Il dispatcher della pagina indirizzerà automaticamente una pagina pubblicata. Di solito la proprietà *Published* viene utilizzata durante la generazione di un menu con collegamenti ipertestuale ad altre pagine. Le pagine che hanno *Published* impostato a *False* non sono elencate nel menu.

Title: testo, lettura

Vedere anche: Esempio 5 (page B-22), Esempio 18 (page B-35)

Fornisce il titolo della pagina.

Il titolo viene di solito reso visibile all'utente.

Oggetti globali

Gli oggetti globali possono essere referenziati dagli script lato server; negli script è possibile utilizzare riferimenti agli oggetti globali che somigliano ai riferimenti agli oggetti nel codice sorgente. Ad esempio:

```
<%= Application.Title %>
```

visualizza il titolo dell'applicazione in una pagina web.

Gli oggetti globali di script sono elencati di seguito:

Tabella B.2 Oggetti globali WebSnap

Oggetto	Descrizione
Oggetto Application (page B-15)	Accede ai campi e alle azioni dell'adapter dell'applicazione, come il campo Title.
Oggetto EndUser (page B-16)	Accede ai campi e alle azioni dell'adapter dell'utente, come DisplayName dell'utente, l'azione Login e l'azione Logout.
Oggetto Modules (page B-16)	Referenzia un modulo di dati o di pagina tramite nome. Le variabili Modules possono essere utilizzate anche per elencare i moduli dell'applicazione.
Oggetto Page (page B-17)	Accede alle proprietà della pagina che viene generata come la pagina Title.
Oggetto Pages (page B-17)	Referenzia una pagina registrata tramite nome. Le variabili Pages possono essere utilizzate anche per elencare le pagine registrate dell'applicazione.

Tabella B.2 Oggetti globali WebSnap

Oggetto	Descrizione
Oggetto Producer (page B-17)	Scrive il contenuto HTML che può includere marcatori trasparenti.
Oggetto Request (page B-18)	Accede alle proprietà e ai metodi della richiesta HTTP.
OggettoResponse (page B-18)	Scrive dei contenuti HTML nella risposta HTTP.
Oggetto Session (page B-19)	Accede alle proprietà della sessione dell'utente.

Oggetto Application

Vedere anche: Tipo Adapter (page B-2)

L'oggetto Application permette l'accesso alle informazioni sull'applicazione.

Utilizzare l'oggetto Application per accedere ai campi e alle azioni dell'adapter dell'applicazione, come il campo *Title*. L'oggetto Application è un Adapter, e può quindi essere personalizzato con campi e azioni aggiuntivi. È possibile accedere ai campi e alle azioni aggiunti all'adapter dell'applicazione tramite nome come proprietà dell'oggetto Application.

Proprietà

Designing: Booleano, lettura

Vedere anche: Esempio 1 (page B-20)

Indica se l'applicazione web viene progettata nell'IDE.

Utilizzare la proprietà *Designing* per generare in modo condizionale del codice HTML che deve essere differente in fase di progettazione rispetto a quello della fase di esecuzione dell'applicazione web.

ModulePath: testo, lettura

Vedere anche: metodo *QualifyFileName*

Identifica la posizione dell'applicazione web eseguibile.

Utilizzare *ModulePath* per costruire nomi di file che si trovano nella stessa direcoty del file eseguibile.

ModuleFileName: testo, lettura

Vedere anche: metodo *QualifyFileName*

Identifica il nome per esteso del file dell'eseguibile.

Title: testo, lettura

Vedere anche: Esempio 18 (page B-35)

Fornisce il titolo dell'applicazione.

La proprietà *Title* ha il valore della proprietà *TApplicationAdapter*. Di solito questo valore viene visualizzato all'inizio delle pagine HTML.

Metodi

QualifyFileName(FileName): testo

Vedere anche: Esempio 1 (page B-20)

Converte un nome di file relativo o un riferimento di directory in un riferimento assoltuto.

QualifyFileName utilizza la posizione della directory dell'eseguibile dell'applicazione web per qualificare un nome di file che non è per specificato per esteso. Il metodo restituisce un nome di file per esteso. Se il parametro FileName è per esteso, il nome del file viene restituito invariato. In fase di progettazione, il parametro FileName viene qualificato con l'ubicazione della directory del file di progetto.

Oggetto EndUser

Vedere anche: Adapter type (page B-2)

Permette l'accesso a informazioni riguardanti l'utente corrente.

Utilizzare *EndUser* per accedere ai campi e alle azioni dell'adapter per utente finale, come *DisplayName* dell'utente. È possibile accedere ai campi e alle azioni che sono stati aggiunti all'adapter per utente finale tramite nome come proprietà dell'oggetto *EndUser*.

Proprietà

DisplayName: testo, lettura

Vedere anche: Esempio 19 (page B-35)

Fornisce il nome dell'utente.

LoggedIn: Booleano, lettura

Vedere anche: Esempio 19 (page B-35)

Indica se l'utente è connesso.

LoginFormAction:Tipo AdapterAction, lettura

Vedere anche: Esempio 19 (page B-35), tipo AdapterAction(page B-4)

Fornisce l'azione dell'adapter utilizzataper connettere un utente.

LogoutAction: tipo AdapterAction , lettura

Vedere anche: Esempio 19 (page B-35), tipo AdapterAction(page B-4)

Fornisce l'azione dell'adapter utilizzata per disconnettere un utente.

Oggetto Modules

Vedere anche: Esempio 2 (page B-21), Esempio 20 (page B-36)

L'oggetto Modules permette l'accesso a tutti i moduli che sono stati istanziati o attivati per soddisfare la richiesta HTTP corrente.

Per referenziare un particolare modulo utilizzare il nome del modulo come proprietà della variabile Modules. Per elencare tutti i moduli all'interno dell'applicazione, creare un enumeratore utilizzando l'oggetto Modules.

Oggetto Page

Vedere anche: Esempio 5 (page B-22), tipo Page (page B-13)

L'oggetto Page permette l'accesso alle proprietà della pagina che viene generata.

Vedere il tipo Page per una descrizione delle proprietà e dei metodi dell'oggetto Page.

Oggetto Pages

Vedere anche: Esempio 5 (page B-22)

L'oggetto Pages permette l'accesso a tutte le pagine registrate dall'applicazione.

Per referenziare una pagina particolare utilizzare il nome della pagina come proprietà della variabile Pages. Per elencare tutte le pagine all'interno dell'applicazione, creare un enumeratore utilizzando l'oggetto Pages.

Oggetto Producer

Vedere anche: oggetto Response (page B-18)

Utilizzare l'oggetto Producer per scrivere del testo contenente marcatori trasparenti. I marcatori saranno tradotti dal produttore di pagine e quindi scritti nella risposta HTTP. Se il testo non contiene marcatori trasparenti, utilizzare l'oggetto Response per migliorare le prestazioni.

Proprietà

Content: testo, lettura/scrittura

Permette l'accesso alla parte dei contenuti della risposta HTTP.

Utilizzare *Content* per leggere o scrivere l'intera parte dei contenuti della risposta HTTP. L'impostazione a *Content* permette la traduzione dei marcatori trasparenti. Se non si utilizzano marcatori trasparenti, impostare invece Response.*Content* per migliorare le prestazioni.

Metodi

Write(Value)

Esegue un'aggiunge alla parte dei contenuti della richiesta HTTP con supporto per i marcatori trasparenti.

Utilizzare il metodo *Write* per eseguire un'aggiunta alla parte dei contenuti della richiesta HTTP. Il metodo *Write* traduce marcatori trasparenti come:

```
Write('Translate this: <#MyTag>')
```

Se non si utilizzano marcatori trasparenti, utilizzare Response.*Write* per migliorare le prestazioni.

Oggetto Request

Fornisce accesso alla richiesta HTTP.

Utilizza le proprietà dell'oggetto Response per accedere alle informazioni sulla richiesta HTTP.

Proprietà

Host: testo, lettura

Riporta il valore dello header Host della richiesta HTTP.

Host è uguale alla proprietà *Host* di *TWebRequest*.

PathInfo: testo, lettura

Contiene la parte PathInfo dell'URL.

PathInfo è uguale alla proprietà *InternalPathInfo* di *TWebRequest*.

ScriptName: testo, lettura

Contiene la parte del nome dello script dell'URL, che specifica il nome di un'applicazione per Web server.

ScriptName è uguale alla proprietà *InternalScriptName* di *TWebRequest*.

Oggetto Response

Vedere anche: oggetto Producer(page B-17)

Permette l'accesso alla risposta HTTP. Utilizzare l'oggetto Response per scrivere nella parte dei contenuti della risposta HTTP. Se si utilizzano marcatori trasparenti, utilizzare l'oggetto Producer invece dell'oggetto Response.

Proprietà

Content: testo, lettura/scrittura

Fornisce accesso alla parte dei contenuti della risposta HTTP.

Utilizzare *Content* per leggere o scrivere l'intera parte dei contenuti della risposta HTTP.

Metodi

Write(Value)

Vedere anche: Esempio 5 (page B-22)

Aggiunge Value alla parte del contenuto della richiesta HTTP.

Utilizzare il metodo *Write* per aggiungere Value al contenuto della richiesta HTTP. Il metodo *Write* non traduce i marcatori trasparenti.

Utilizzare il metodo *Write* dell'oggetto Producer per scrivere una stringa che contiene uno o più marcatori trasparenti.

Oggetto Session

L'oggetto Session permette l'accesso all'ID e ai valori della sessione.

Un oggetto Session è utilizzato per tenere traccia delle informazioni sull'utente per un breve periodo di tempo

Proprietà

SessionID.Value: testo, lettura/scrittura

Permette l'accesso all'ID della sessione dell'utente corrente.

Values(Name): variant, lettura

Permette l'accesso ai valori conservati nella sessione corrente dell'utente.

Esempi JScript

I seguenti esempi JScript dimostrano quali sono le proprietà e i metodi utilizzati dello scripting lato server.

Tabella B.3 Esempi JScript di scripting lato server

Esempio	Descrizione
Esempio 1 (page B-20)	Utilizza il metodo *QualifyFilename* dell'oggetto Application per generare un riferimento di percorso relativo a un'immagine.
Esempio 2 (page B-21)	Dichiara una variabile che referenzia un modulo.
Esempio 3 (page B-21)	Elenca i moduli nell'applicazione web e visualizza i loro nomi in una tabella HTML.
Esempio 4 (page B-21)	Dichiara una variabile che referenzia una pagina registrata.
Esempio 5 (page B-22)	Enumera le pagine registrate per generare un menu di collegamenti ipertestuali alle pagine pubblicate.
Esempio 6 (page B-22)	Enumera le pagine registrate per generare un menu di collegamenti ipertestuali alle azioni predefinite delle pagine pubblicate.
Esempio 7 (page B-23)	Scrive un elenco di errori rilevati da un adapter.
Esempio 8 (page B-23)	Enumera tutti gli oggetti azione di un adapter per visualizzare i valori delle proprietà dell'oggetto azione in una tabella HTML.
Esempio 9 (page B-24)	Enumera i record di un adapter per visualizzare i valori dei campi dell'adapter in una tabella HTML.

Tabella B.3 Esempi JScript di scripting lato server (continua)

Esempio	Descrizione
Esempio 10 (page B-25)	Genera una scheda HTML per modificare i campi dell'adapter e inoltrare le azioni dell'adapter.
Esempio 11 (page B-26)	Mostra le azioni dell'adapter per supportare l'impaginazione.
Esempio 12 (page B-28)	Mostra l'immagine di un campo dell'adapter utilizzando il marcatore <img>
Esempio 13 (page B-28)	Mostra un campo dell'adapter utilizzando i marcatori <select> e <option>.
Esempio 14 (page B-29)	Mostra un campo dell'adapter come un gruppo di caselle di controllo.
Esempio 15 (page B-31)	Mostra i valori di un campo dell'adapter utilizzando i marcatori e .
Esempio 16 (page B-32)	Mostra un campo dell'adapter come un gruppo di pulsanti di opzione.
Esempio 17 (page B-34)	Utilizza le proprietà DisplayStyle, InputStyle andViewMode del campo dell'adapter per generare HTML.
Esempio 18 (page B-35)	Utilizza le proprietà dell'oggetto Application e dell'oggetto Page per generare un'intestazione di pagina.
Esempio 19 (page B-35)	Utilizza le proprietà dell'oggetto EndUser per mostrare il nome dell'utente, il comando di login e il comando di logout.
Esempio 20 (**page B-36**)	Enumera gli oggetti in un modulo in grado di gestire script.
Esempio 21 (page B-36)	Utilizza la proprietà DisplayStyle delle azioni dell'adapter per generare HTML.
Esempio 22 (page B-37)	Generare una tabella HTML per aggiornare più record di dettaglio.

Esempio 1

Vedere anche: Proprietà *Designing* e *QualifyFileName* dell'oggetto Application (page B-15), proprietà *PathInfo* dell'oggetto Request (page B-18)

Questo esempio genera un riferimento di percorso relativo a un'immagine. Se lo script è in fase di progettazione referenzia un elenco effettivo; altrimenti referenzia un elenco virtuale.

```
<%
     function PathInfoToRelativePath(S)
     {
       var R = '';
       var L = S.length
       I = 0
       while (I < L)
       {
         if (S.charAt(I) == '/')
           R = R + '../'
         I++
       }
       return R
     }
```

```
    function QualifyImage(S)
    {
      if (Application.Designing)
        return Application.QualifyFileName("..\\images\\" + S);   // relative directory
      else
        return PathInfoToRelativePath(Request.PathInfo) + '../images/' + S;  // virtual
directory
    }
  %>
```

Esempio 2

Vedere anche: oggetto Modules (page B-16)

Questo esempio dichiara una variabile che referenzia WebModule1:

```
<% var M  = Modules.WebModule1 %>
```

Esempio 3

Vedere anche: oggetto Modules (page B-16)

L'esempio 3 enumera i moduli istanziati e ne mostra in una tabella il nome della variabile e il nome della classe :

```
<table border=1>
<tr><th>Name</th><th>ClassName</th></tr>
<%
   var e = new Enumerator(Modules)
   for (; !e.atEnd(); e.moveNext())
   {
%>
  <tr><td><%=e.item().Name_%></td><td><%=e.item()ClassName._%></td></tr>
<%
   }
%>
</table>
```

Esempio 4

Vedere anche: oggetto Pages (page B-17), proprietà *Title* dell'oggetto Page (page B-17)

Questo esempio dichiara una variabile che referenzia una pagina di nome Home. Mostra anche il titolo di Home.

```
<% var P = Pages.Home %>
<p><%= P.Title %></p>
```

Esempio 5

Vedere anche: oggetto Pages (page B-17), proprietà *Published* e *HREF*dell'oggetto Page (page B-17), metodo *Write* dell'oggetto Response (page B-18)

Questo esempio enumera le pagine registrate e crea un menu visualizzando collegamenti ipertestuali a tutte le pagine pubblicate.

```
<table>
<td>
<%   e = new Enumerator(Pages)
     s = ''
     c = 0
     for (; !e.atEnd(); e.moveNext())
     {
       if (e.item().Published)
       {
         if (c>0) s += ' | '
         if (Page.Name != e.item().Name)
           s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
         else
           s += e.item().Title
         c++
       }
     }
     if (c>1) Response.Write(s)
%>
</td>
</table>
```

Esempio 6

Vedere anche: proprietà *DefaultAction* del tipo *DefaultAction*(page B-13)

Questo esempio enumera le pagine registrate e crea un menu che mostra collegamenti ipertestuali alle azioni predefinite.

```
<table>
<td>
<%  e = new Enumerator(Pages)
     s = ''
     c = 0
     for (; !e.atEnd(); e.moveNext())
     {
       if (e.item().Published)
       {
         if (c>0) s += ' | '
         if (Page.Name != e.item().Name)
           if (e.item().DefaultAction != null)
             s += '<a href="'+e.item().DefaultAction.AsHREF+'"> +e.item().Title+'</a>'
           else
             s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
         else
```

```
          s += e.item().Title
        C++
      }
    }
    if (c>1) Response.Write(s)
%>
</td>
</table>
```

Esempio 7

Vedere anche: proprietà *Errors* del tipo Adapter (page B-2), tipo AdapterErrors (page B-6), oggetto Modules (page B-16), metodo *Write* dell'oggetto Response (page B-18)

Questo esempio scrive un elenco di errori rilevati da un adapter.

```
<% {
   var e = new Enumerator(Modules.CountryTable.Adapter.Errors)
   for (; !e.atEnd(); e.moveNext())
   {
     Response.Write("<li>" + e.item().Message)
   }
   e.moveFirst()
   } %>
```

Esempio 8

Vedere anche: proprietà *Actions* del tipo Adapter (page B-2), tipo AdapterAction (page B-4)

Questo esempio enumera tutte le azioni di un adapter e mostra in una tabella i valori delle proprietà delle azioni.

```
<% // Display some properties of an action in a table.
   function DumpAction(A)
   {
%>
      <table border="1">
         <tr><th COLSPAN=2><%=A.Name%></th>
         <tr><th>AsFieldValue:</th><td><%= A.AsFieldValue %></td>
         <tr><th>AsHREF:</th><td><%= A.AsHREF %></span>
         <tr><th>DisplayLabel:</th><td><%= A.DisplayLabel %></td>
         <tr><th>Enabled:</th><td><%= A.Enabled %></td>
         <tr><th>CanExecute:</th><td><span class="value"><%= A.CanExecute %></td>
     </table>
<%
  }
%>

<% // Call the DumpAction function for every action in an adapter.
   function DumpActions(A)
   {
```

```
      var e = new Enumerator(A)
      for (; !e.atEnd(); e.moveNext())
      {
        DumpAction(e.item())
      }
    }
%>


<%
// Display properties of actions in the adapter named Adapter1.
DumpActions(Adapter1.Actions) %>
```

Esempio 9

Vedere anche: proprietà *Records* del tipo Adapter (page B-2), proprietà *DisplayText* del tipo AdapterField (page B-7)

Questo esempio genera una tabella HTML enumerando i record di un adapter.

```
<%
// Define variables for the adapter and fields.

vAdapter=Modules.CountryTable.Adapter
vAdapter_Name=vAdapter.Name
vAdapter_Capital=vAdapter.Capital
vAdapter_Continent=vAdapter.Continent
%>

<%
// Function to write column text so that all cells have borders.
function WriteColText(t)
{
   Response.Write((t!="")?t:" ")
}
%>

<table border="1">
  <tr>
  <th>Name</th>
  <th>Capital</th>
  <th>Continent</th>
<%
   // Enumerate all the records in the adapter and write the field values.

   var e = new Enumerator(vAdapter.Records)
   for (; !e.atEnd(); e.moveNext())
   { %>
      <tr>
      <td><div><% WriteColText(vAdapter_Name.DisplayText) %></div></td>
      <td><div><% WriteColText(vAdapter_Capital.DisplayText) %></div></td>
      <td><div><% WriteColText(vAdapter_Continent.DisplayText) %></div></td>
      </tr>
<%
```

```
    }
%>
</table>
```

Esempio 10

Vedere anche: proprietà *LinkToPage* e *ìAsFieldValue* del tipo AdapterAction (page B-4), proprietà *InputName* e *DisplayText* del tipo AdapterField (page B-7), proprietà *HiddenFields* e *HiddenRecordFields* del tipo Adapter (page B-2)

Questo esempio genera una scheda HTML per modificare i campi dell'adapter e inoltra le azioni dell'adapter.

```
<%
// Define some variables for the adapter, fields, and actions.

vAdapter=Modules.CountryTable.Adapter
vAdapter_Name=vAdapter.Name
vAdapter_Capital=vAdapter.Capital
vAdapter_Continent=vAdapter.Continent
vAdapter_Apply=vAdapter.Apply
vAdapter_RefreshRow=vAdapter.RefreshRow

// Put the adapter in Edit mode unless the mode is already set.  If the mode is already
// set this is probably because an adapter action set the mode. For example, an insert
// row action would put the adapter in Insert mode.

if (vAdapter.Mode=="")
  vAdapter.Mode="Edit"
%>
<form name="AdapterForm1" method="post">

  <!-- This hidden field defines the action that is executed when the form is submitted. -->

  <input type="hidden" name="__act">


<%
   // Write hidden fields defined by the adapter.

   if (vAdapter.HiddenFields != null)
   {
     vAdapter.HiddenFields.WriteFields(Response)
   } %>
<% if (vAdapter.HiddenRecordFields != null)
   {
     vAdapter.HiddenRecordFields.WriteFields(Response)
   } %>
  <table>
    <tr>
      <td>
        <table>
          <tr>
```

```
            <!-- Write input fields to edit the fields of the adapter -->

            <td>Name</td>
            <td ><input type="text" size="24" name="<%=vAdapter_Name.InputName%>" value="
               <%= vAdapter_Name.EditText %>" ></td>
            </tr>
            <tr>
            <td>Capital</td>
            <td ><input type="text" size="24" name="<%=vAdapter_Capital.InputName%>"
               value="<%= vAdapter_Capital.EditText %>" ></td>
            </tr>
            <tr>
            <td>Continent</td>
            <td ><input type="text" size="24" name="<%=vAdapter_Continent.InputName%>"
               value="<%= vAdapter_Continent.EditText %>" ></td>
            </tr>
          </table>
        </td>
      </tr>
      <tr>
        <td>
          <table>
            <!-- Write submit buttons to execute actions.  Use LinkToPage so this
                 page is regenerated after executing an action. -->

            <tr>
            <td><input type="submit" value="Apply"
              onclick = "AdapterForm1.__act.value='
                <%=vAdapter_Apply.LinkToPage(Page.Name).AsFieldValue%>'"></td>
            <td><input type="submit" value="Refresh"
              onclick = "AdapterForm1.__act.value='
                <%=vAdapter_RefreshRow.LinkToPage(Page.Name).AsFieldValue%>'"> </td>
            </tr>
          </table>
        </td>
      </tr>
    </table>
  </form>
```

Esempio 11

Vedere anche: proprietà *Array* e *AsHREF* del tipo AdapterAction (page B-4)

Questo esempio mostra le azioni dell'adapter per supportare l'impaginazione. Le azioni PrevPage, GotoPage e NextPage vengono mostrate come collegamenti ipertestuali. L'azione GotoPage ha un array di comandi. I comandi vengono enumerati in modo da generare un collegamento ipertestuale per passare ad ogni singola pagina.

```
<%
   // Define variables for the adapter and actions.

   vAdapter = Modules.WebDataModule1.QueryAdapter
```

```
   vPrevPage = vAdapter.PrevPage
   vGotoPage = vAdapter.GotoPage
   vNextPage = vAdapter.NextPage
%>

<!-- Generate a table that displays hyperlinks between pages. -->

<table cellpadding="5">
<tr>
<td>
<%
   // Prevpage displays "<<".  Use an anchor tag only if the command is enabled.

   if (vPrevPage.Enabled)
   { %>
     <a href="<%=vPrevPage.LinkToPage(Page.Name).AsHREF%>"><<</a>
<%
   }
   else
   {%>
     <a><<</a>
 <%} %>
<%
   // GotoPage has a list of commands. Loop through the list.
   // Use an anchor tag only if the command is enabled.

   if (vGotoPage.Array != null)
   {
     var e = new Enumerator(vGotoPage.Array)
     for (; !e.atEnd(); e.moveNext())
     {
%>
       <td>
<%     if (vGotoPage.Enabled)
       { %>
          <a href="<%=vGotoPage.LinkToPage(Page.Name).AsHREF%>">
            <%=vGotoPage.DisplayLabel%></a>
<%     }
       else
       { %>
          <a><%=vGotoPage.DisplayLabel%></a>
<%     }
%>
       </td>
<%
     }
   }
%>
<td>
<%
   // NextPage displays ">>".  Use an anchor tag only if the command is enabled.

   if (vNextPage.Enabled)
   { %>
```

```
      <a href="<%=vNextPage.LinkToPage(Page.Name).AsHREF%>">>></a>
<%
   }
   else
   {%>
      <a>>></a>
 <%} %>
</td>
</table>
```

Esempio 12

Vedere anche: proprietà *Image* del tipo AdapterField (page B-7)

L'esempio 12 mostra l'immagine di un campo dell'adapter.

```
<%
// Declare variables for the adapter and field.

vAdapter=Modules.WebDataModule3.DataSetAdapter1
vGraphic=vAdapter.Graphic
%>

<!-- Display the adapter field as an image. -->
<img src="<%=(vGraphic.Image!=null) ? vGraphic.Image.AsHREF : ''%>"
alt="<%=vGraphic.DisplayText%>">
```

Esempio 13

Vedere anche: proprietà *Values* e *ValuesList* del tipo AdapterField (page B-7)

Questo esempio scrive un campo dell'adapter con elementi HTML <select> e <option>.

```
<%
// Return an object that defines HTML select options for an adapter field.
// The returned object has the following elements:
//
// text - string containing the <option> elements.
// count - the number of <option> elements.
// multiple - string containing the either 'multiple' or ''.
//            Use this value as an attribute of the <select> element.
//
// Use as follows:
//   obj=SelOptions(f)
//   Response.Write('<select size="' + obj.count + '" name="' + f.InputName + '" ' +
//     obj.multiple + '>' + obj.text + '</select>')

function SelOptions(f)
{
   var s=''
   var v=''
   var n=''
```

```
    var c=0
    if (f.ValuesList != null)
    {
      var e = new Enumerator(f.ValuesList.Records)
      for (; !e.atEnd(); e.moveNext())
      {
        s+= '<option'
        v = f.ValuesList.Value;
        var selected
        if (f.Values == null)
           selected = f.IsEqual(v)
        else
           selected = f.Values.HasValue(v)
        if (selected)
           s += ' selected'
        n = f.ValuesList.ValueName;
        if (n=='')
        {
          n = v
          v = ''
        }
        if (v!='') s += ' value="' + v + '"'
        s += '>' + n + '</option>\r\n'
        c++
      }
      e.moveFirst()
    }
    r = new Object;
    r.text = s
    r.count = c
    r.multiple = (f.Values == null) ? '' : 'multiple'
    return r;
  }
%>

<%
   // Generate HTML select options for an adapter field
   function WriteSelectOptions(f)
   {
      obj=SelOptions(f)
%>
      <select size="<%=obj.count%>" name="<%=f.InputName%>" <%=obj.multiple%> >
        <%=obj.text%>
      </select>
<%
   }
%>
```

Esempio 14

Vedere anche: proprietà *Values* e *ValuesList* del tipo AdapterField (page B-7)

Questo esempio scrive un campo dell’adapter come gruppo di elementi <input type=”checkbox”>.

```
<%
// Return an object that defines HTML check boxes for an adapter field.
// The returned object has the following elements:
//
// text - string containing the <input type=checkbox> elements.
// count - the number of <option> elements.
//
// Use as follows to define a check box group with three columns and no additional
// attributes:
//   obj=CheckBoxGroup(f, 3, ’’)
//   Response.Write(obj.text)
//
function CheckBoxGroup(f,cols,attr)
{
   var s=’’
   var v=’’
   var n=’’
   var c=0;
   var nm=f.InputName
   if (f.ValuesList == null)
   {
     s+= ’<input type="checkbox"’
     if (f.IsEqual(true)) s+= ’ checked’
     s += ’ value="true"’ + ’ name="’ + nm + ’"’
     if (attr!=’’) s+= ’ ’ + attr
     s += ’></input>\r\n’
     c = 1
   }
   else
   {
     s += ’<table><tr>’
     var e = new Enumerator(f.ValuesList.Records)
     for (; !e.atEnd(); e.moveNext())
     {
       if (c % cols == 0 && c != 0) s += ’</tr><tr>’
       s+= ’<td><input type="checkbox"’
       v = f.ValuesList.Value;
       var checked
       if (f.Values == null)
         checked = (f.IsEqual(v))
       else
         checked = f.Values.HasValue(v)
       if (checked)
         s+= ’ checked’
       n = f.ValuesList.ValueName;
       if (n==’’)
         n = v
       s += ’ value="’ + v + ’"’ + ’ name="’ + nm + ’"’
       if (attr!=’’) s+= ’ ’ + attr
       s += ’>’ + n + ’</input></td>\r\n’
       c++
```

```
      }
      e.moveFirst()
      s += '</tr></table>'
    }
    r = new Object;
    r.text = s
    r.count = c
    return r;
  }
%>

<%
   // Write an adapter field as a check box group
   function WriteCheckBoxGroup(f, cols, attr)
   {
      obj=CheckBoxGroup(f, cols, attr)
      Response.Write(obj.text);
   }
%>
```

Esempio 15

Vedere anche: proprietà *Values* e *ValuesList* del tipo AdapterField (page B-7), tipo AdapterFieldValues (page B-10)

Questo esempio scrive un campo dell'adapter come elenco di valori a sola lettura utilizzando gli elementi <ul> e <li>.

```
<%
// Return an object that defines HTML list values for an adapter field.
// The returned object has the following elements:
//
// text - string containing the <li> elements.
// count - the number of elements.
//
// text will be blank and count will be zero if the adapter field does not
// support multiple values.
//
// Use as follows to define a displays a read only list of this an adapter
// fields values.
//   obj=ListValues(f)
//   Response.Write('<ul>' + obj.text + '</ul>')
//
function ListValues(f)
{
   var s=''
   var v=''
   var n=''
   var c=0;
   r = new Object;
   if (f.Values != null)
   {
     var e = new Enumerator(f.Values.Records)
```

```
      for (; !e.atEnd(); e.moveNext())
      {
        s+= '<li>'
        s += f.Values.ValueField.DisplayText;
        s += '</li>'
        c++
      }
      e.moveFirst()
    }
    r.text = s
    r.count = c
    return r;
}
%>

<%
// Write an adapter field as a list of read-only values.
function WriteListValues(f)
{
   obj=ListValues(f)
%>
   <ul><%=obj.text%></ul>
<%
}
%>
```

Esempio 16

Vedere anche: Tipo AdapterFieldValuesList (page B-11), proprietà *Values* e *ValuesList* del tipo AdapterField (page B-7)

Questo esempio scrive un campo dell'adapter come un gruppo di elementi < input type="radio">.

```
<%
// Return an object that defines HTML radio buttons for an adapter field.
// The returned object has the following elements:
//
// text - string containing the <input type=radio> elements.
// count - the number of elements.
//
// Use as follows to define a radio button group with three columns and no additional
// attributes:
//   obj=RadioGroup(f, 3, '')
//   Response.Write(obj.text)
//

function RadioGroup(f,cols,attr)
{
   var s=''
   var v=''
   var n=''
   var c=0;
```

```
    var nm=f.InputName
    if (f.ValuesList == null)
    {
      s+= '<input type="radio"'
      if (f.IsEqual(true)) s+= ' checked'
      s += ' value="true"' + ' name="' + nm + '"'
      if (attr!='') s+= ' ' + attr
      s += '></input>\r\n'
      c = 1
    }
    else
    {
      s += '<table><tr>'
      var e = new Enumerator(f.ValuesList.Records)
      for (; !e.atEnd(); e.moveNext())
      {
        if (c % cols == 0 && c != 0) s += '</tr><tr>'
        s+= '<td><input type="radio"'
        v = f.ValuesList.Value;
        var checked
        if (f.Values == null)
          checked = (f.IsEqual(v))
        else
          checked = f.Values.HasValue(v)
        if (checked)
          s+= ' checked'
        n = f.ValuesList.ValueName;
        if (n=='')
        {
        n = v
        }
        s += ' value="' + v + '"' + ' name="' + nm + '"'
        if (attr!='') s+= ' ' + attr
        s += '>' + n + '</input></td>\r\n'
        c++
      }
      e.moveFirst()
      s += '</tr></table>'
    }
    r = new Object;
    r.text = s
    r.count = c
    return r;
  }
%>

<%
   // Write an adapter field as a radio button group
   function WriteRadioGroup(f, cols, attr)
   {
      obj=RadioGroup(f, cols, attr)
      Response.Write(obj.text);
   }
%>
```

Esempio 17

Vedere anche: AdapterField type's *DisplayStyle, ViewMode*, and *InputStyle* properties (page B-7)

Questo esempio genera del codice HTML per un campo dell'adapter in base ai valori delle proprietà *InputStyle*, *DisplayStyle* e *ViewMode*.

```
<%
   function WriteField(f)
   {
     Mode = f.ViewMode
     if (Mode == 'Input')
     {
       Style = f.InputStyle
       if (Style == 'SelectMultiple' || Style == 'Select')
          WriteSelectOptions(f)
       else if (Style == 'CheckBox')
         WriteCheckBoxGroup(f, 2, '')
       else if (Style == 'Radio')
         WriteRadioGroup(f, 2, '')
       else if (Style == 'TextArea')
       {
%>
         <textarea wrap=OFF name="<%=f.InputName%>"><%= f.EditText %></textarea>
<%
       }
       else if (Style == 'PasswordInput')
       {
%>
         <input type="password" name="<%=f.InputName%>"/>
<%
       }
       else if (Style == 'File')
       {
%>
          <input type="file" name="<%=f.InputName%>"/>
<%
       }
       else
       {
%>
         <input type="input" name="<%=f.InputName%>" value="<%= f.EditText %>"/>
<%
       }
     }
     else
     {
       Style = f.DisplayStyle
       if (Style == 'List')
         WriteListValues(f)
       else if (Style == 'Image')
       {
%>
```

```
          <img src="<%=(f.Image!=null) ? f.Image.AsHREF : ''%>" alt="<%=f.DisplayText%>">
<%      }
        else
          Response.Write('<p>' + f.DisplayText + '</p>')
      }
    }
%>
```

Esempio 18

Vedere anche: oggetto Page (page B-17), proprietà *Title* dell'oggetto Application (page B-15)

Questo esempio utilizza proprietà dell'oggetto Application e dell'oggetto Page per generare un'intestazione di pagina.

```
<html>
<head>
<title>
<%= Page.Title %>
</title>
</head>
<body>
<h1><%= Application.Title %></h1>

<h2><%= Page.Title %></h2>
```

Esempio 19

Vedere anche: oggetto EndUser (page B-16)

Questo esempio utilizza le proprietà dell'oggetto EndUser per visualizzare il nome, il comando di login e il comando di logout di un utente.

```
<% if (EndUser.Logout != null)
   {
     if (EndUser.DisplayName != '')
     {
%>
       <h1>Welcome <%=EndUser.DisplayName %></h1>
<%   }
     if (EndUser.Logout.Enabled) {
%>
       <a href="<%=EndUser.Logout.AsHREF%>">Logout</a>
<%   }
     if (EndUser.LoginForm.Enabled) {
%>
       <a href="<%=EndUser.LoginForm.AsHREF%>">Login</a>
<%   }
   }
%>
```

Esempio 20

Vedere anche: Tipo Module (page B-12)

Questo esempio elenca gli oggetti in un modulo in grado di gestire script .

```
<%
   // Write an HTML table list the name and classname of all scriptable objects in a module.
   function ListModuleObjects(m)
   {
%>
     <p></p>
     <table border="1">
     <tr>
     <th colspan="2"><%=m.Name_ + ': ' + m.ClassName_%></th>
     </tr>
<%
     var e = new Enumerator(m.Objects)
     for (; !e.atEnd(); e.moveNext())
     {
%>
     <tr>
     <td>
        <%= e.item().Name_ + ': ' + e.item().ClassName_ %>
     </td>
     </tr>
<%
     }
%>
     </table>
<%
   }
%>
```

Esempio 21

Vedere anche: proprietà *DisplayStyle* e *Enabled* del tipo AdapterAction (page B-4)

Questo esempio genera del codice HTML per un'azione dell'adapter in base alla proprietà *DisplayStyle* dell'azione.

```
<%
   // Write HTML for an adapter action using the DisplayStyle property.
   //
   // a - action
   // cap - caption.  If blank the action's display label is used.
   // fm - name of the HTML form
   // p - page name to goto after action execution.  If blank, the current page is used.
   //
   // Note that this function does not use the action's Array property.   Is is assumed that
   // the action has a single command.
   //
   function WriteAction(a, cap, fm, p)
   {
```

```
        if (cap == '')
          cap = a.DisplayLabel
        if (p == '')
          p = Page.Name
        Style = a.DisplayStyle
        if (Style == 'Anchor')
        {

           if (a.Enabled)
           {
             // Do not use the href property.  Instead, submit the form so that  HTML form
             // fields are part of the HTTP request.
%>
             <a href=""onclick="<%=fm%>.__act.value='
                <%=a.LinkToPage(p).AsFieldValue%>';<%=fm%>.submit();return false;">
                <%=cap%></a>
<%
           }
           else
           {
%>
             <a><%=cap%></a>
<%
           }
        }
        else
        {
%>
           <input type="submit" value="<%= cap%>"
onclick="<%=fm%>.__act.value='<%=a.LinkToPage(p).AsFieldValue%>'">
<%
        }
    }
%>
```

Esempio 22

Vedere anche: proprietà *HiddenFields*, *HiddenRecordFields,* e *Mode* del tipo Adapter (page B-2)

Questo esempio genera una tabella HTML per aggiornare più record di dettaglio.

```
<%
vItemsAdapter=Modules.DM.ItemsAdapter
vOrdersAdapter=Modules.DM.OrdersAdapter
vOrderNo=vOrdersAdapter.OrderNo
vCustNo=vOrdersAdapter.CustNo
vPrevRow=vOrdersAdapter.PrevRow
vNextRow=vOrdersAdapter.NextRow
vRefreshRow=vOrdersAdapter.RefreshRow
vApply=vOrdersAdapter.Apply
vItemNo=vItemsAdapter.ItemNo
vPartNo=vItemsAdapter.PartNo
vDiscount=vItemsAdapter.Discount
```

```
vQty=vItemsAdapter.Qty
%>

<!-- Use two adapters to update multiple detail records.
     The orders adapter is associated with the master dataset.
     The items adapter is associated with the detail dataset.
     Each row in a grid displays values from the items adapter.  One
     column display an <input> element for editing Qty.  The apply button
     updates the Qty value in each detail record.-->

<!-- Display the order number and customer number values. -->
<h2>OrderNo: <%= vOrderNo.DisplayText %></h2>
<h2>CustNo: <% vCustNo.DisplayText %></h2>

<%
   // Put the items adapter in edit mode because this form updates
   // the Qty field.
   vItemsAdapter.Mode = 'Edit'
%>

<form name="AdapterForm1" method="post">

  <!-- Define a hidden field for submitted the action name and parameters -->
  <input type="hidden" name="__act">

<%
   // Write hidden fields containing state information about the
   // orders adapter and items adapter.
   if (vOrdersAdapter.HiddenFields != null)
     vOrdersAdapter.HiddenFields.WriteFields(Response)
   if (vItemsAdapter.HiddenFields != null)
     vItemsAdapter.HiddenFields.WriteFields(Response)

   // Write hidden fields containing state information about the current
   // record of the orders adapter.
   if (vOrdersAdapter.HiddenRecordFields != null)
     vOrdersAdapter.HiddenRecordFields.WriteFields(Response)%>

  <table border="1">
    <tr>
    <th>ItemNo</th>
    <th>PartNo</th>
    <th>Discount</th>
    <th>Qty</th>
  </tr>
<%
   var e = new Enumerator(vItemsAdapter.Records)
   for (; !e.atEnd(); e.moveNext())
   { %>
     <tr>
     <td><%=vItemNo.DisplayText%></td>
     <td><%=vPartNo.DisplayText%></td>
     <td><%=vDiscount.DisplayText%></td>
     <td><input type="text" name="<%=vQty.InputName%>" value="<%= vQty.EditText %>" ></td>
```

```
     </tr>
<%
     // Write hidden fields containing state information about each record of the
     // items adapter. This is needed by the items adapter when updating the Qty field.

     if (vItemsAdapter.HiddenRecordFields != null)
       vItemsAdapter.HiddenRecordFields.WriteFields(Response)
   }
%>
  </table>
  <p></p>
  <table>
    <td><input type="submit" value="Prev Order"

onclick="AdapterForm1.__act.value='<%=vPrevRow.LinkToPage(Page.Name).AsFieldValue%>'"></td>
    <td><input type="submit" value="Next Order"

onclick="AdapterForm1.__act.value='<%=vNextRow.LinkToPage(Page.Name).AsFieldValue%>'"></td>
    <td><input type="submit" value="Refresh"

onclick="AdapterForm1.__act.value='<%=vRefreshRow.LinkToPage(Page.Name).AsFieldValue%>'"></
td>
    <td><input type="submit" value="Apply"
      onclick="AdapterForm1.__act.value='<%=vApply.LinkToPage(Page.Name).AsFieldValue%>'"></
td>
  </table>
</form>
```

Indice

Simboli

Numeri

A

B

C

D

E

F

G

H

I

J

K

L

N

O

P

Q

R

S

T

U

V

W

X

Y